Cambridge IGCSE

Chuyên luyện thi IELTS/SAT → AP / IB / A-levels

8020 ENGLISH

Cambridge IGCSE Computer Science

Giáo trình luyện thi chuyên sâu

Song ngữ Anh–Việt

8020 ENGLISH

5A/2 Trần Phú, P.4, Q.5, TP.HCM · luyenthi8020.com

Thầy Tường 0332 747 169 · Cô Nancy 0983 422 692

Lời mở đầu · Foreword

Cuốn sách này thuộc bộ giáo trình luyện thi chương trình quốc tế của **8020 English (8020 Education)** — trung tâm luyện thi trọng điểm **IELTS · SAT · AP · IB · A-level · IGCSE** và săn học bổng du học. Toàn bộ nội dung được biên soạn bởi đội ngũ **Giáo sư · Tiến sĩ · Thạc sĩ** tốt nghiệp các trường top đầu thế giới, bám sát khung chương trình chính thức, trình bày đầy đủ lý thuyết, ví dụ mẫu, hình minh hoạ và hệ thống bài tập có lời giải chi tiết.

This book is part of the 8020 English international exam-prep series: complete English theory with Vietnamese explanations, worked examples, illustrations and fully-solved practice, aligned to the official framework.

Thành tích 8020 English

9.0 IELTS cao nhất	**1570** SAT cao nhất
5/5 AP — điểm tuyệt đối	**90/100** IB Diploma
100% GV $\geq$ 8.0 IELTS / 1500 SAT	**CMU** Tiến sĩ ĐH #1 Hoa Kỳ về CS

Đội ngũ tiêu biểu: Thầy Châu Cát Tường (Academic Head, ThS Western Sydney, top 1% IELTS 8.5) · TS Nguyễn Anh Huy (Carnegie Mellon, cựu kỹ sư Amazon) · TS Nguyễn Phước Trường (Toán, ĐH Klagenfurt) · cùng đội ngũ giảng viên SAT 1500–1600, IELTS 8.0–8.5.

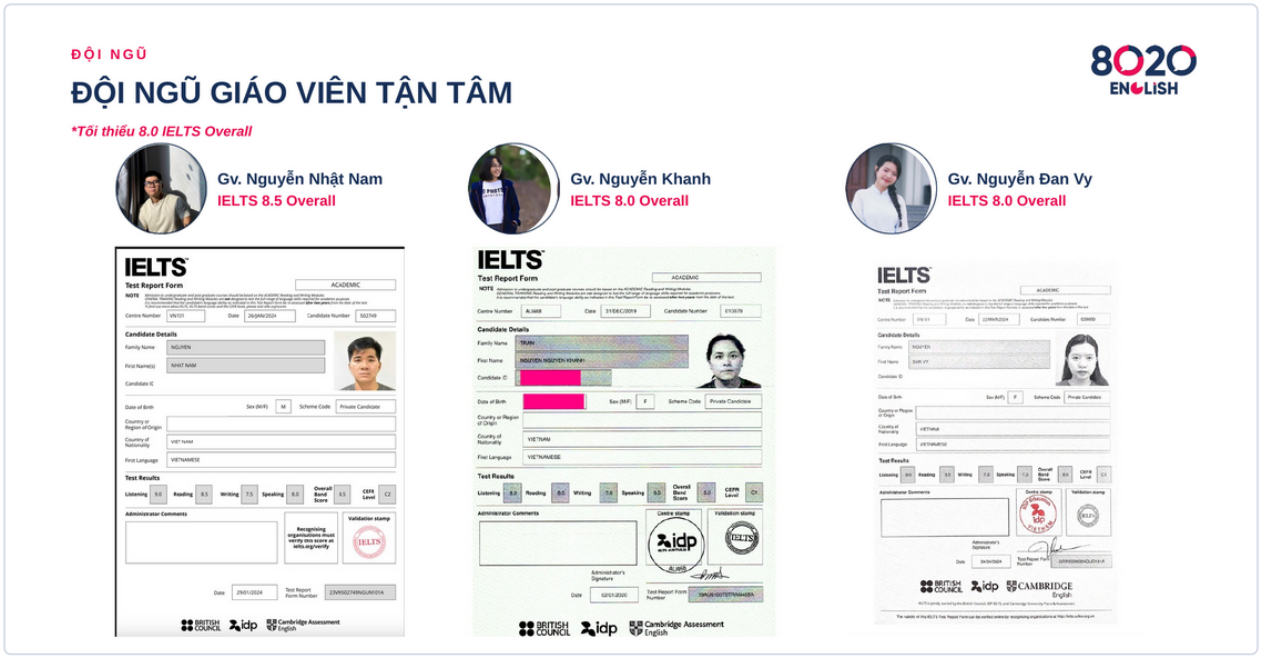
Kết quả học viên — điểm thi thật: IELTS 8.0–9.0 · SAT 1550 / 1570 / 1590 · AP 5/5 (Calculus BC, Microeconomics) · IB 90/100 (Economics) · Duolingo 110 · PTE 90 · TOEFL 120. **Bảng vàng SAT:** Khánh Đăng 1510 · Thảo Nguyên 1520 · Tâm Anh 1530.

"Cuối cùng đã đậu vào trường top như mong muốn." — Phan Thị Thanh Hằng, IELTS Overall 8.5.

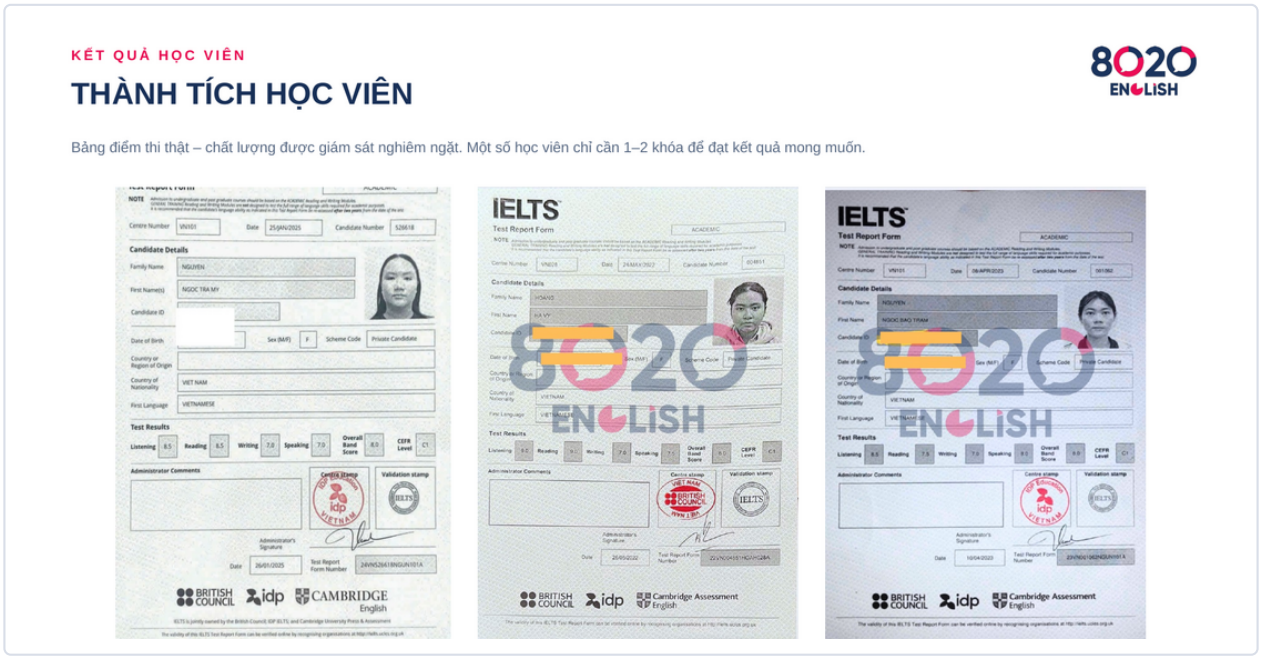
Cách dùng sách hiệu quả: mỗi Unit gồm (1) **Lý thuyết trọng tâm** tiếng Anh kèm **giải thích tiếng Việt**; (2) **Ví dụ mẫu** có lời giải; (3) **Hệ thống bài tập** kèm **lời giải chi tiết**. Hãy tự làm bài trước khi xem lời giải để đạt hiệu quả tối đa.

THÀNH TÍCH THỰC TẾ · 8020 ENGLISH

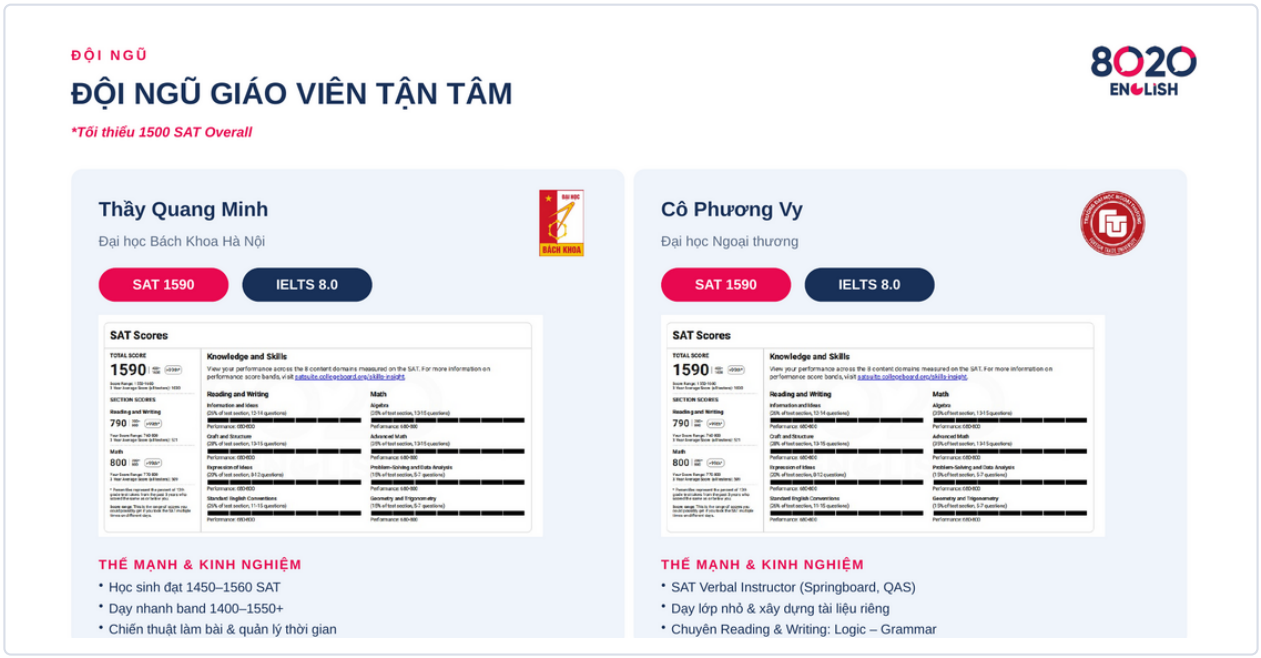
Điểm thi thật · Bằng & bảng điểm thật của giảng viên và học viên 8020 English — **Điểm thật · Thầy thật · Kết quả thật**



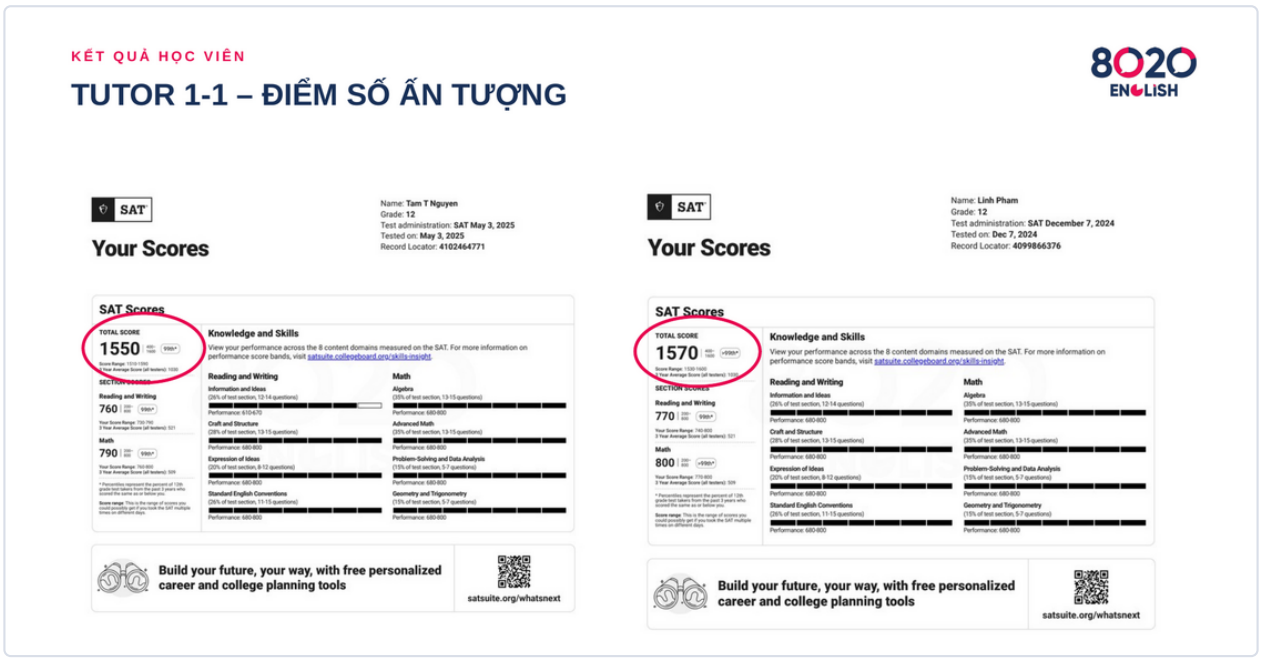
Giảng viên 8020 — IELTS 8.5 & 8.0 Overall (British Council · IDP · Cambridge)



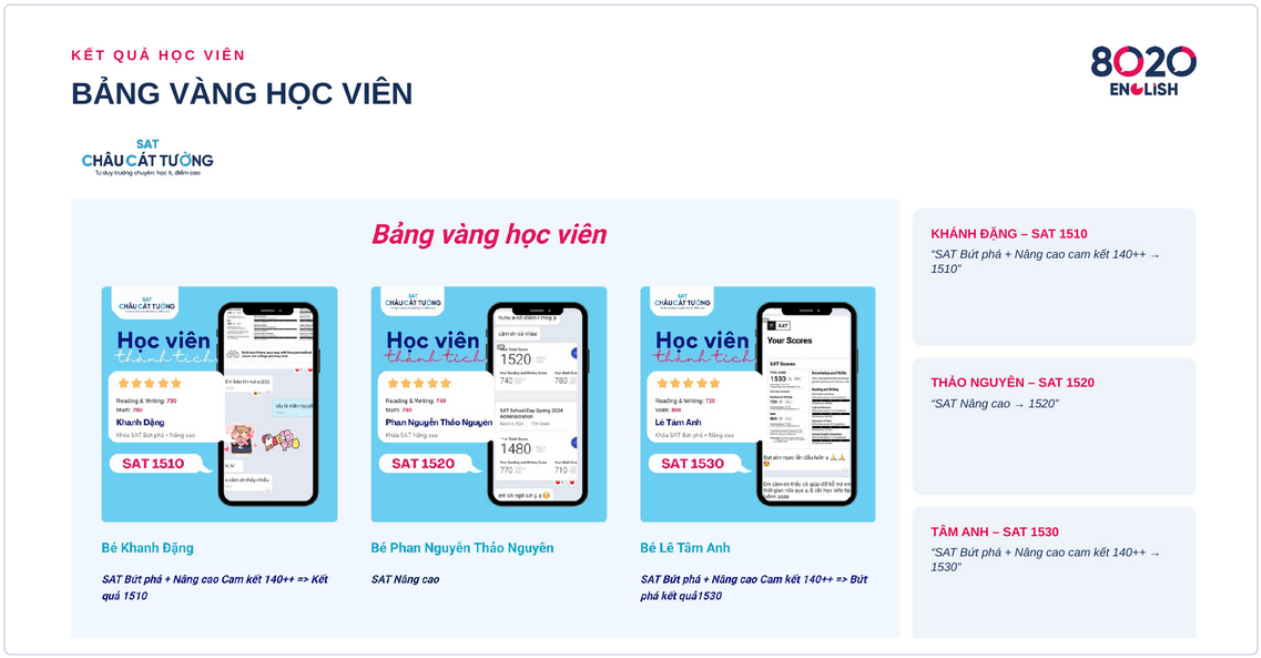
Học viên 8020 — IELTS 8.0 Overall (bảng điểm thật)



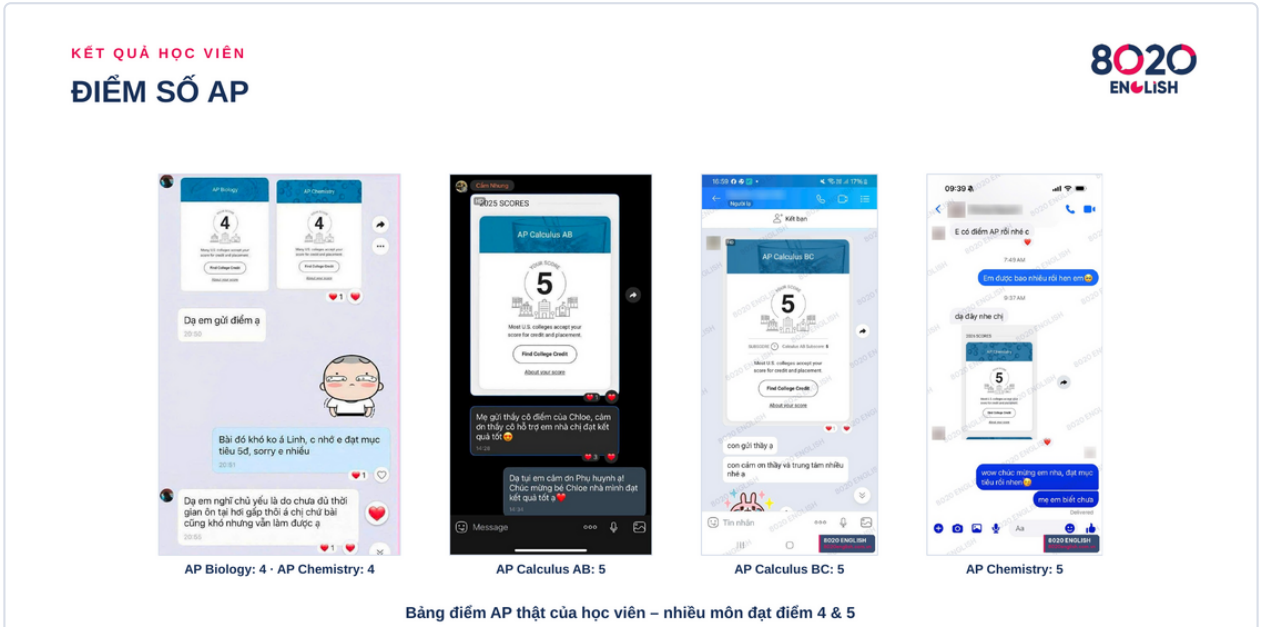
Giảng viên 8020 — SAT 1590 (ĐH Bách Khoa Hà Nội & ĐH Ngoại thương)



Học viên 8020 — SAT 1550 & 1570 (College Board)



Bảng vàng SAT — Học viên đạt 1510 · 1520 · 1530



Điểm AP thật — Calculus AB/BC 5/5 · Biology & Chemistry 4–5 (College Board)

Mục lục · Contents

CAMBRIDGE IGCSE • Unit 1

Data Representation

Mục tiêu Unit: Hệ nhị phân, hệ thập lục phân (hex), phép cộng nhị phân và hiện tượng tràn số (overflow), số âm bù hai (two's complement) và phép cộng bù hai, bảng mã ký tự (ASCII/Unicode), đơn vị lưu trữ dữ liệu, biểu diễn ảnh và âm thanh số, và các phương pháp nén dữ liệu (nén mất dữ liệu/không mất dữ liệu).

Every value a computer stores or moves — a number, a letter, a photograph, a song, a video — is ultimately held as a sequence of **bits** (binary digits, 0 or 1), because digital circuits are built from transistors that reliably distinguish only two electrical states (on/off, high/low voltage). This chapter builds up, layer by layer, how those raw bit patterns are interpreted as numbers, negative numbers, text, images and sound, how many bits different kinds of data actually need, and how that footprint can be reduced through compression.

1.1 Number systems: denary, binary and hexadecimal

1.1.1 Place value and binary–denary conversion

In everyday **denary** (base 10) numbers, each column has a place value that is a power of 10 (..., 1000, 100, 10, 1). **Binary** (base 2) works identically, except each column's place value is a power of 2. In an 8-bit binary number, the place values from left (most significant bit) to right (least significant bit) are $2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0$, i.e. $128, 64, 32, 16, 8, 4, 2, 1$.

Khái niệm cốt lõi

Binary → denary: add together the place values of every column that contains a 1.
Denary → binary: repeatedly find the largest power of 2 that is $\leq$ the remaining value, place a 1 in that column and subtract it, then move to the next column (place a 0 if the power does not fit); or equivalently, divide repeatedly by 2 and read the remainders from bottom to top.

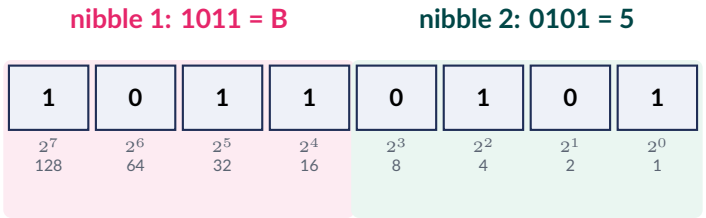


Figure 1.1. 8-bit pattern `10110101`: sum the columns with a 1 $\Rightarrow 128 + 32 + 16 + 4 + 1 = 181$. Grouped into 4-bit nibbles, `1011`=`B` and `0101`=`5`, giving hex `B5`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mỗi cột nhị phân có **giá trị vị trí** là luỹ thừa của 2. Đổi **nhị phân sang thập phân**: cộng các giá trị vị trí có bit 1. Đổi **thập phân sang nhị phân**: trừ dần luỹ thừa của 2 lớn nhất còn vừa (đặt bit 1), hoặc chia liên tiếp cho 2 rồi đọc số dư từ dưới lên.

Ví dụ mẫu · Worked example

Convert the denary number 233 to binary and to hexadecimal.

Largest power of 2 that fits: 128 (remainder 105) $\rightarrow$ 64 (remainder 41) $\rightarrow$ 32 (remainder 9) $\rightarrow$ 16 does not fit (0) $\rightarrow$ 8 (remainder 1) $\rightarrow$ 4 does not fit (0) $\rightarrow$ 2 does not fit (0) $\rightarrow$ 1 (remainder 0). So $233 = 11101001$. Check: $128 + 64 + 32 + 8 + 1 = 233$. ☐

Hex: split into nibbles $1110\ 1001 = \mathrm{E}\ 9 = \mathrm{E9}$. Check: $E9_{16} = 14 \times 16 + 9 = 224 + 9 = 233$. ☐

Ví dụ mẫu · Worked example

Convert the denary number 179 to binary using the **repeated division by 2** method.

Divide by 2 repeatedly, recording the remainder each time: $179 \div 2 = 89$ r1; $89 \div 2 = 44$ r1; $44 \div 2 = 22$ r0; $22 \div 2 = 11$ r0; $11 \div 2 = 5$ r1; $5 \div 2 = 2$ r1; $2 \div 2 = 1$ r0; $1 \div 2 = 0$ r1. Reading the remainders from the *last* one obtained back to the *first* (bottom to top): 10110011. Check by place value: $128 + 32 + 16 + 2 + 1 = 179$. ☐ Both conversion methods — subtracting powers of 2, or dividing repeatedly by 2 — always agree; use whichever is faster for you.

1.1.2 How many bits are needed?

A closely related question is: given a set of N distinct values that must each be stored, what is the *smallest* number of bits required?

Khái niệm cốt lõi

An n-bit pattern can represent exactly 2^n distinct values. To store N distinct values, find the smallest n such that $2^n \geq N$. Equivalently: keep doubling from $2^1 = 2$ until the running total first reaches or exceeds N; the number of doublings taken is the number of bits needed.

Ví dụ mẫu · Worked example

A quiz app must store scores that can be any whole number from 0 to 59 inclusive (60 possible values). How many bits are needed to store one score?

$2^5 = 32$, which is less than 60 (not enough patterns). $2^6 = 64$, which is ≥ 60. So the smallest suitable n is $\boxed{6 \text{ bits}}$ (giving 64 possible patterns, of which 60 are actually used).

Ví dụ mẫu · Worked example

A sensor must store any whole number from 0 to 1000 inclusive. How many bits are needed?

The range 0 to 1000 inclusive contains $1000 - 0 + 1 = 1001$ distinct values. $2^9 = 512$, which is less than 1001 (not enough). $2^{10} = 1024$, which is ≥ 1001. So $\boxed{10 \text{ bits}}$ are required.

(!) Lỗi thường gặp: When a range is described as "0 to X inclusive", remember it contains $X+1$ distinct values, not X — forgetting to add 1 is a common source of an answer that is one bit too small.

VN **GIẢI THÍCH TIẾNG VIỆT**

Với n bit, ta biểu diễn được đúng 2^n giá trị khác nhau. Để lưu N giá trị khác nhau, cần tìm n nhỏ nhất sao cho $2^n \geq N$. Lưu ý: khoảng "từ 0 đến X" (bao gồm cả hai đầu) có $X + 1$ giá trị, không phải X — quên cộng thêm 1 là lỗi khiến tính thiếu 1 bit.

1.1.3 Hexadecimal shorthand

Hexadecimal (base 16, digits `0`–`9` then `A`–`F` representing 10–15) is used throughout computing because each hex digit maps *exactly* onto a 4-bit **nibble** ($2^4 = 16$ possible nibble values, matching the 16 hex digits). This makes long binary strings — memory addresses, MAC addresses, colour codes — far shorter and less error-prone for humans to read, write and type than the equivalent string of 0s and 1s.

Ví dụ mẫu · Worked example

Convert `10110101` to hexadecimal, and confirm both equal 181 in denary.

Split into nibbles: `1011 0101` = `B 5`, so the value is `B5` in hex. Check: $B5_{16} = (11 \times 16) + 5 = 176 + 5 = 181$ — matches $128 + 32 + 16 + 4 + 1 = 181$ found by binary place value. □

Ví dụ mẫu · Worked example

Convert the hexadecimal value `2F` to binary and to denary.

Each hex digit expands to its own 4-bit nibble: `2` = `0010`, `F` = `1111`. So `2F` = `00101111` in binary (8 bits, since 2 hex digits = 2 nibbles = 8 bits). Denary: $2 \times 16 + 15 = 32 + 15 = 47$. Check by place value: $32 + 8 + 4 + 2 + 1 = 47$. □

Ví dụ mẫu · Worked example

Convert the 16-bit binary pattern `1111000010101100` to hexadecimal and denary.

Split into four nibbles: `1111 0000 1010 1100` = `F 0 A C`, so the hex value is `F0AC`. Denary from hex: $(15 \times 4096) + (0 \times 256) + (10 \times 16) + 12 = 61440 + 0 + 160 + 12 = \boxed{61612}$. Check by binary place value (place values $32768, \ldots, 1$): $32768 + 16384 + 8192 + 4096 + 128 + 32 + 8 + 4 = 61612$. □

(!) Lỗi thường gặp: Always split binary into nibbles *from the right*. If the number of bits is not a multiple of 4, pad extra zeros on the **left** first (e.g. `101` has 3 bits, so pad to `0101` = `5`) — padding on the right would silently change the value.

1.1.4 Binary addition and overflow

Binary addition works exactly like denary column addition: add each column together with any **carry** from the column to its right, writing down the result bit and carrying 1 into the next column whenever a column's total reaches 2 or more (since binary has no digit for "2").

Khái niệm cốt lõi

A register (or a byte in memory) can only hold a **fixed number of bits**. If the true result of an addition needs *more* bits than the register has, the extra carry-out bit is simply lost, and the value stored is **wrong** — this is called **overflow**. For an n-bit unsigned register, overflow occurs whenever the true sum is $\geq 2^n$.

Ví dụ mẫu · Worked example

Add the 8-bit binary numbers `01101011` (107) and `00111001` (57), showing carries.

```
carry: 1 1 1 1 1 0 1 0
```

```
      0 1 1 0 1 0 1 1   (107)
+     0 0 1 1 1 0 0 1   (57)
----------------------
      1 0 1 0 0 1 0 0   (164)
```

Working from the right: $1+1=10$ (write 0, carry 1); $1+0+1=10$ (write 0, carry 1); $0+0+1=1$ (write 1); $1+1=10$ (write 0, carry 1); $0+1+1=10$ (write 0, carry 1); $1+1+1=11$ (write 1, carry 1); $1+0+1=10$ (write 0, carry 1); $0+0+1=1$ (write 1). Result: `10100100`. Check: $128+32+4=164=107+57$. ✓ The final carry-out is 0, so no overflow: 164 fits comfortably in 8 bits (max 255).

Ví dụ mẫu · Worked example

Add the 8-bit unsigned binary numbers `11001000` (200) and `01100100` (100), and comment on the result.

Adding column by column produces `00101100`, with a final carry-*out* of 1 that does not fit in the 8-bit register and is discarded. Reading only the retained 8 bits: $00101100=32+8+4=44$. But the true sum is $200+100=300$, which needs 9 bits ($100101100_2=256+32+8+4=300$). Because the register can only store 8 bits, the leading 1 is lost and the stored result is the wrong value 44 (in fact $300-256=44$) — this is **overflow**: the true result did not fit in the available register width.

(!) Lỗi thường gặp: Overflow is *not* the same as a wrong calculation method — the addition itself is done correctly bit by bit. The error happens purely because the **register is too narrow** to hold the true answer, so the most significant carry-out bit is discarded. Always check whether the true denary sum exceeds the maximum value the register can represent (2^n-1 for an unsigned n-bit register) before trusting a binary addition result.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cộng nhị phân thực hiện giống hệt cộng thập phân theo cột, có **nhớ (carry)** sang cột kế tiếp khi tổng cột ≥ 2. **Tràn số (overflow)** xảy ra khi kết quả thật sự cần **nhiều bit hơn** số bit mà thanh ghi (register) có thể lưu — bit nhớ cuối cùng bị mất, khiến giá trị lưu lại **sai**. Với thanh ghi n bit không dấu, tràn số xảy ra khi tổng thật $\geq 2^n$.

1.2 Two's complement

Computers represent **negative integers** using **two's complement**, so that addition and subtraction can be performed by the *same* binary-adder circuitry used for positive numbers, with no special-case logic for signs. In an n-bit two's complement number, the most-significant (leftmost) bit has a *negative* place value (-2^{n-1}), while every other bit keeps its usual positive place value. The representable range is -2^{n-1} to $2^{n-1}-1$ (for 8 bits: -128 to 127).

To negate a two's complement number: write the positive binary value, **invert every bit** (this step alone is called the *ones' complement*), then **add 1**. Applying the same two-step procedure a second time converts back to the original positive value — negation is its own inverse.

Ví dụ mẫu · Worked example

Represent -73 in 8-bit two's complement.
$73 = 01001001$. Invert all bits: 10110110. Add 1: 10110111. So $-73 = 10110111$ (the leading 1 signals a negative value).

Ví dụ mẫu · Worked example

What denary value does the 8-bit two's complement pattern 11101011 represent?
The leading bit is 1, so the value is negative. Invert: 00010100; add 1: $00010101 = 21$. So the pattern represents -21. Check directly by place value ($-128, 64, 32, 16, 8, 4, 2, 1$): $-128 + 64 + 32 + 8 + 2 + 1 = -21$. □

Ví dụ mẫu · Worked example

What denary value does the 8-bit two's complement pattern 10001101 represent?
Leading bit is 1 (negative). Invert: 01110010; add 1: $01110011 = 64+32+16+2+1 = 115$. So the pattern represents -115. Check directly by negative place value: $-128+8+4+1 = -115$. □

VN **GIẢI THÍCH TIẾNG VIỆT**

Bù hai (two's complement): đảo tất cả các bit của số dương rồi **cộng thêm 1** để ra số âm tương ứng; làm ngược lại (đảo bit rồi cộng 1) để đổi từ số âm về số dương. Với n bit, phạm vi biểu diễn là -2^{n-1} đến $2^{n-1}-1$. Bit đầu tiên (trái nhất) mang **giá trị vị trí âm** (-2^{n-1}).

1.2.1 Adding two's complement numbers

The great advantage of two's complement is that a positive number and a negative number can be added using *ordinary* binary addition on their bit patterns — there is no need to "subtract" separately. Any carry out of the leftmost (sign) bit is simply discarded, exactly as with unsigned addition.

Ví dụ mẫu · Worked example

Add 45 and -30 in 8-bit two's complement, using ordinary binary addition.
$45 = 00101101$. For -30: $30 = 00011110$, invert $\rightarrow 11100001$, add 1 $\rightarrow 11100010$.

```
   0 0 1 0 1 1 0 1   (45)
+  1 1 1 0 0 0 1 0   (-30)
-----------------------
1  0 0 0 0 1 1 1 1   (discard the leading carry-out)
```

Retained 8 bits: $00001111 = 8+4+2+1 = 15$. Leading bit is 0 (positive), so the answer is $+15$ — matching $45 + (-30) = 15$ exactly. □ The carry out of the sign bit column is simply thrown away, and the result is still correct because the true answer fits comfortably within the 8-bit range.

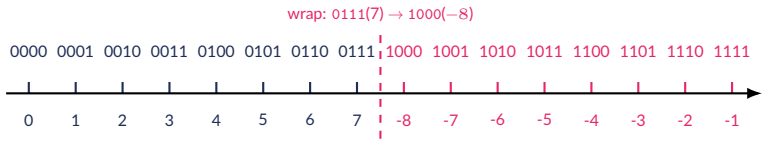


Figure 1.2. All sixteen 4-bit two's complement patterns. Values 0 to 7 use patterns 0000–0111; values -8 to -1 use patterns 1000–1111. The pattern one greater than 0111 (i.e. 1000) does *not* continue to $+8$: it wraps around to the most negative value, -8.

1.2.2 Detecting overflow in two's complement addition

Because a two's complement register has a fixed range (-2^{n-1} to $2^{n-1}-1$), adding two numbers can produce a true mathematical result that does not fit — again called **overflow**. There is a simple rule to spot it:

Khái niệm cốt lõi

Overflow rule: in n-bit two's complement addition, overflow has occurred if and only if the two numbers being added have the *same* sign bit, but the result has the *opposite* sign bit. (Adding a positive and a negative number can *never* overflow, because the true result must lie between the two operands, well within range.)

Ví dụ mẫu · Worked example

Add 100 and 90 in 8-bit two's complement and check for overflow.
$100 =$ 01100100, $90 =$ 01011010 (both positive: sign bit 0).

```
   0 1 1 0 0 1 0 0   (100)
+  0 1 0 1 1 0 1 0   (90)
-----------------------
   1 0 1 1 1 1 1 0
```

Result 10111110 has sign bit 1 (appears negative). Decoding it: invert → 01000001, add 1 → 01000010$= 66$, so the pattern reads as -66. Both operands were positive (sign bit 0), but the result's sign bit is 1 — by the overflow rule, this **is overflow**. The true sum $100 + 90 = 190$ exceeds the maximum positive value an 8-bit two's complement register can hold (127), so the sign bit is corrupted ($190 - 256 = -66$, confirming the wraparound).

Ví dụ mẫu · Worked example

Add -70 and -80 in 8-bit two's complement and check for overflow.
$-70 =$ 10111010, $-80 =$ 10110000 (both negative: sign bit 1).

```
   1 0 1 1 1 0 1 0   (-70)
+  1 0 1 1 0 0 0 0   (-80)
-----------------------
1  0 1 1 0 1 0 1 0
```

Discarding the leading carry-out, the retained result is 01101010, sign bit 0 (appears positive) $= 64 + 32 + 8 + 2 = 106$. Both operands were negative (sign bit 1), but the result's sign bit is 0 — **overflow has occurred**. The true sum $-70 + (-80) = -150$ is more negative than the minimum an 8-bit register can hold (-128), so the stored value wraps to $-150 + 256 = 106$, an incorrect apparent positive result.

(!) Lỗi thường gặp: Do not assume overflow just because a carry is generated out of the sign-bit column — that carry is discarded routinely, even in perfectly correct additions (see the $45 + (-30)$ example above). Overflow is detected by comparing the **sign bits of the operands** against the **sign bit of the result**, not by looking at the carry alone.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cộng số dương với số âm ở dạng bù hai chỉ cần **cộng nhị phân bình thường**, bỏ qua bit nhớ tràn ra khỏi bit dấu. **Tràn số (overflow)** trong phép cộng bù hai xảy ra khi **hai số hạng cùng dấu** nhưng **kết quả lại khác dấu** — nghĩa là kết quả thật vượt quá phạm vi biểu diễn của thanh ghi n bit (-2^{n-1} đến $2^{n-1}-1$). Cộng một số dương với một số âm thì **không bao giờ** gây tràn số.

1.3 Character sets

A **character set** assigns a unique binary code to every character a computer can store, display or transmit. **ASCII** (American Standard Code for Information Interchange) uses 7 bits per character, giving $2^7 = 128$ possible codes — just enough for uppercase and lowercase English letters, digits 0–9, common punctuation, and a handful of control codes (e.g. carriage return, tab). **Extended ASCII** uses the full 8 bits of a byte, giving $2^8 = 256$ codes, with the extra 128 codes used for additional symbols, accented letters or box-drawing characters — but which extra symbols occupy which extra codes was never fully standardised, so different systems could interpret the top 128 codes of an extended ASCII byte differently. Such a mapping of byte values to specific characters is called a **code page**; if a file created using one code page is opened by software expecting a different code page, the codes beyond the basic 128 can be displayed as the *wrong* characters (a form of corrupted text sometimes nicknamed "mojibake").

Unicode solves this by using many more bits per character (commonly 16, or a variable number of bytes in encodings such as UTF-8), giving well over a million possible code points. This is large enough to give a single, unique, universally agreed code to characters from virtually *every* written language on Earth (English, Vietnamese with all its diacritics, Chinese, Japanese, Arabic, Cyrillic…) plus mathematical symbols and emoji — something ASCII's 128 or extended ASCII's 256 codes could never fit, and without the code-page ambiguity of extended ASCII.

VN **GIẢI THÍCH TIẾNG VIỆT**

ASCII dùng 7 bit ($2^7 = 128$ mã) cho chữ cái/số/ký hiệu tiếng Anh cơ bản. **Extended ASCII** dùng đủ 8 bit (256 mã), nhưng 128 mã bổ sung không được chuẩn hoá thống nhất giữa các hệ thống (gọi là **code page** khác nhau), nên mở sai code page có thể hiển thị **sai ký tự**. **Unicode** dùng nhiều bit hơn hẳn nên biểu diễn được **rất nhiều ngôn ngữ khác nhau** (kể cả tiếng Việt có dấu) và biểu tượng cảm xúc (emoji) một cách thống nhất — điều mà ASCII không đủ chỗ để làm.

1.4 Data storage units

Every quantity of digital storage is built from the bit upward: a **nibble** is 4 bits (one hex digit); a **byte** is 8 bits (2 nibbles, typically one ASCII character); larger units are named kilobyte, megabyte, gigabyte, terabyte, and so on.

> For Cambridge IGCSE calculations, storage units follow the **binary convention**: 1 KB $= 1024$ bytes $= 2^{10}$ bytes, and each larger unit is 1024 times the one below it.

Unit	Size	As a power of 2	Typical content
Bit	1 binary digit	2^0	one 0 or 1
Nibble	4 bits	2^2 bits	one hexadecimal digit
Byte	8 bits	2^3 bits	one ASCII character
Kilobyte (KB)	1024 bytes	2^{10} bytes	a short text file
Megabyte (MB)	1024 KB	2^{20} bytes $= 1{,}048{,}576$ B	a photograph
Gigabyte (GB)	1024 MB	2^{30} bytes	a movie file
Terabyte (TB)	1024 GB	2^{40} bytes	a large hard drive

Figure 1.3. Exam calculations use the binary convention shown above. Storage-*device* marketing, however, sometimes instead uses **decimal** SI prefixes (1 KB $= 1000$ B, 1 GB $= 10^9$ B) — which is one reason a drive labelled "500 GB" shows only about $500 \times 10^9 \div 2^{30} \approx 465.7$ GB once a computer measures it using the binary convention.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đơn vị lưu trữ theo quy ước **nhị phân** dùng trong đề thi: 1 nibble $= 4$ bit, 1 byte $= 8$ bit, 1 KB $= 1024$ byte, 1 MB $= 1024$ KB, 1 GB $= 1024$ MB, 1 TB $= 1024$ GB. Nhà sản xuất ổ cứng đôi khi dùng quy ước **thập phân** (1 GB $= 10^9$ byte) khiến dung lượng hiển thị trên máy tính (đo theo nhị phân) **thấp hơn** con số quảng cáo.

1.5 Representing images

A bitmap image is stored as a rectangular grid of **pixels** (picture elements). Its **resolution** is the number of pixels it contains, expressed as width $\times$ height. Each pixel's colour is stored as a binary code whose length — the **colour depth** — determines how many distinct colours are possible: a colour depth of d bits per pixel allows 2^d possible colours (e.g. 8 bits $\rightarrow$ 256 colours; 24 bits, "true colour", $\rightarrow 2^{24} \approx 16.7$ million colours).

Khái niệm cốt lõi

Uncompressed bitmap file size (in bits) = width (pixels) $\times$ height (pixels) $\times$ colour depth (bits per pixel). Increasing resolution *or* colour depth increases file size — both are independent multipliers.

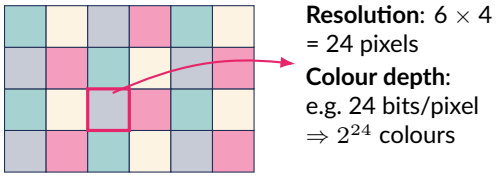


Figure 1.4. A small 6×4-pixel image. Each highlighted pixel stores a binary colour code **colour depth** bits long; more bits per pixel allow more distinct colours but increase file size (size in bits = width $\times$ height $\times$ colour depth).

Ví dụ mẫu · Worked example

An uncompressed bitmap image is 320 pixels wide, 240 pixels tall, with a colour depth of 24 bits (true colour). Find the file size in KB (1 KB $= 1024$ bytes).

Size in bits $= 320 \times 240 \times 24 = 1{,}843{,}200$ bits. In bytes: $1{,}843{,}200 \div 8 = 230{,}400$ bytes. In KB: $230{,}400 \div 1024 = \boxed{225 \text{ KB}}$.

Ví dụ mẫu · Worked example

An image is stored at 640×480 resolution with an 8-bit colour depth. By what factor would the file size increase if the colour depth were changed to 24 bits, and what would the new file size be in KB?

Increasing colour depth from 8 to 24 bits multiplies file size by $24 \div 8 = 3$ (resolution is unchanged). Original size: $640 \times 480 \times 8 = 2{,}457{,}600$ bits $= 307{,}200$ bytes $= 300$ KB exactly. At 24-bit depth: $300 \times 3 = \boxed{900 \text{ KB}}$ (check directly: $640 \times 480 \times 24 = 7{,}372{,}800$ bits $= 921{,}600$ bytes $= 900$ KB ✓).

Colour depth in practice: a **1-bit** depth gives $2^1 = 2$ colours — pure black and white only (a *bitonal* image, used e.g. for simple line-art or fax scans). An **8-bit greyscale** depth gives $2^8 = 256$ shades of grey, from black through to white. **24-bit true colour** is normally split into three 8-bit channels — Red, Green and Blue (**RGB**) — each independently storing 256 possible intensity levels (0–255) for that colour component; combining the three channels gives $256 \times 256 \times 256 = 256^3 = 2^{24} \approx 16.7$ million possible colours.

Ví dụ mẫu · Worked example

A scanner captures a document at 1000×800 pixels using 8-bit **greyscale**. Find the file size in KB, and compare it with storing the same document in 24-bit true colour.
Greyscale: $1000 \times 800 \times 8 = 6{,}400{,}000$ bits $= 800{,}000$ bytes $= 800{,}000 \div 1024 \approx \boxed{781.25 \text{ KB}}$.
True colour: $1000 \times 800 \times 24 = 19{,}200{,}000$ bits $= 2{,}400{,}000$ bytes $= 2{,}400{,}000 \div 1024 \approx 2343.75$ KB. The true-colour version is $2343.75 \div 781.25 = 3$ times larger, since it uses 24 bits per pixel instead of 8 — consistent with storing 3 colour channels instead of a single grey-level channel.

VN **GIẢI THÍCH TIẾNG VIỆT**
Ảnh đen trắng thuần (bitonal): 1 bit/điểm ảnh, chỉ 2 màu. **Ảnh xám (greyscale)** 8-bit: 256 mức xám. **Ảnh màu thật (true colour)** 24-bit thường tách thành 3 kênh **RGB (Đỏ–Lục–Lam)**, mỗi kênh 8 bit (256 mức), kết hợp cho $256^3 = 2^{24} \approx 16{,}7$ triệu màu.

1.5.1 Image file headers and metadata

A raw grid of pixel colour codes is meaningless on its own — software reading the file needs to know *how* to interpret the stream of bits. For this reason, image file formats store a **header** containing **metadata**: values such as the image's width and height (in pixels), its colour depth, and often the compression method used. Without this metadata, an image-viewing program would not know where one row of pixels ends and the next begins, or how many bits make up each pixel's colour — the same raw bits could be misread as the wrong dimensions or wrong colours, producing a distorted, stretched, or unreadable image.

VN **GIẢI THÍCH TIẾNG VIỆT**
Ảnh bitmap là lưới **điểm ảnh (pixel)**; **độ phân giải** là số điểm ảnh (rộng $\times$ cao); **độ sâu màu** là số bit/điểm ảnh, cho $2^{\text{độ sâu màu}}$ màu có thể có. Kích thước file (bit) = rộng $\times$ cao $\times$ độ sâu màu. Phần **tiêu đề (header)** của file ảnh lưu **metadata** (chiều rộng, chiều cao, độ sâu màu…) để phần mềm biết cách **giải mã đúng** luồng bit thành ảnh — thiếu metadata này, ảnh có thể bị đọc sai kích thước hoặc sai màu.

1.6 Representing sound

Sound is a continuous (**analogue**) wave of varying air pressure. To store it digitally, the wave's amplitude is measured, or **sampled**, at regular time intervals, and each measurement is stored as a binary number. Two factors control quality and file size: the **sample rate** (how many samples are taken per second, measured in Hz) and the **sample resolution** (how many bits are used to store each individual sample's amplitude value).

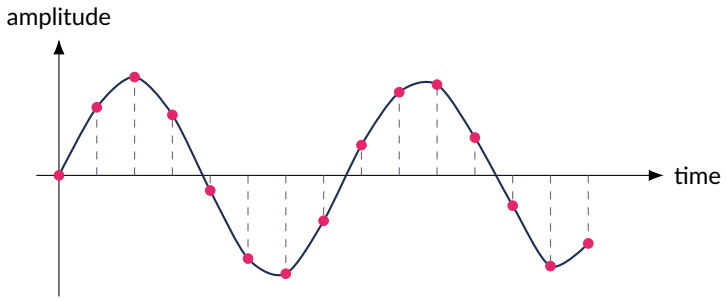


Figure 1.5. The analogue sound wave (curve) is measured at regular time intervals (dashed lines mark **sample points**, dots are the recorded values). The **sample rate** is how often a sample is taken (samples per second, Hz); the **sample resolution** is how many bits store each sample's amplitude.

Khái niệm cốt lõi

Increasing either the sample rate or the sample resolution captures the original analogue wave more accurately (higher **playback quality**), because more, or more precise, snapshots of the wave are stored — but each additional sample, or each additional bit per sample, adds directly to the file size. Sound quality and file size are therefore a direct trade-off.

Ví dụ mẫu · Worked example

A mono audio clip is recorded at a sample rate of $44{,}100$ Hz, sample resolution 16 bits, for a duration of 3 seconds. Find the file size in KB.

Size in bits $= 44{,}100 \times 16 \times 3 = 2{,}116{,}800$ bits. In bytes: $2{,}116{,}800 \div 8 = 264{,}600$ bytes. In KB: $264{,}600 \div 1024 \approx \boxed{258.4 \text{ KB}}$.

Ví dụ mẫu · Worked example

The same clip (44,100 Hz, 16-bit, 3 seconds) is instead recorded in **stereo** (2 channels, i.e. separate left and right samples are stored). Find the new file size in KB, and compare it to the mono version.

Stereo doubles the number of samples stored per second (one set per channel), so size in bits $= 44{,}100 \times 16 \times 3 \times 2 = 4{,}233{,}600$ bits. In bytes: $4{,}233{,}600 \div 8 = 529{,}200$ bytes. In KB: $529{,}200 \div 1024 \approx \boxed{516.8 \text{ KB}}$ — exactly double the mono file size of 258.4 KB, since a second, independent channel of samples must be stored.

VN **GIẢI THÍCH TIẾNG VIỆT**

Âm thanh được ghi bằng cách **lấy mẫu (sampling)** biên độ sóng tương tự (analogue) theo các khoảng thời gian đều nhau. **Tần số lấy mẫu (sample rate)** và **độ phân giải mẫu (sample resolution)** càng cao thì chất lượng phát lại càng tốt nhưng kích thước file cũng càng lớn. Ghi **âm thanh nổi (stereo, 2 kênh)** có kích thước gấp đôi so với **đơn kênh (mono)** với cùng thông số, vì phải lưu thêm một kênh mẫu độc lập.

1.7 Data compression

Uncompressed multimedia files (as calculated above) are often very large, so **compression** algorithms are used to reduce file size for storage or transmission.

Khái niệm cốt lõi

Lossy compression (e.g. JPEG for photos, MP3 for audio) permanently discards some data judged least noticeable to humans, giving much smaller files that *cannot* be restored to the exact original. **Lossless compression** (e.g. **Run-Length Encoding, RLE**) removes redundancy *without* discarding any data, so the original can be perfectly reconstructed — but the size reduction achieved is usually smaller than lossy methods can achieve. RLE replaces a run of identical consecutive values with a single (value, count) pair, so it compresses images with large blocks of one colour very well, but gives little or no benefit on photographs, where colour tends to vary gradually and continuously between neighbouring pixels, leaving few or no long runs of identical values.

Run #	Colour	Run length	Encoded pair
1	White	40	`(W,40)`
2	Black	15	`(B,15)`
3	White	25	`(W,25)`

Figure 1.6. 80 individual pixel values compressed to 3 (value, count) pairs — a *lossless* reduction, since the original row of 80 pixels can be perfectly reconstructed from the 3 pairs.

Ví dụ mẫu · Worked example

RLE encoding. A monochrome bitmap row contains 60 black pixels, then 10 white pixels, then 30 black pixels. Write the run-length encoded representation.

`(B,60)(W,10)(B,30)` — 3 pairs (6 stored values) replace 100 individual pixel values.

Ví dụ mẫu · Worked example

RLE decoding. A monochrome bitmap row has been run-length encoded as `(W,12)(B,3)(W,7)(B,18)`. Reconstruct the original row and state how many pixels it contains.

Expand each pair back into that many repeated pixels, in order: 12 white pixels, then 3 black pixels, then 7 white pixels, then 18 black pixels. Total pixels $= 12 + 3 + 7 + 18 = \boxed{40}$ pixels.

1.7.1 Choosing lossy or lossless compression

Lossy compression is well suited to photographs and music, where the human eye or ear cannot perceive the small details discarded, and a large reduction in file size is valuable (e.g. for streaming or web pages). Lossy compression is, however, **unsuitable** for text documents, program (executable) files, or spreadsheets of financial data: discarding even a single bit could change a word, corrupt a line of program code so it no longer runs correctly, or silently alter a number in a set of accounts. For this kind of data, only **lossless** compression (which guarantees an exact, bit-for-bit original on decompression) is acceptable, even though the resulting size reduction is typically smaller than a lossy method could achieve on the same file.

VN **GIẢI THÍCH TIẾNG VIỆT**

Nén mất dữ liệu (lossy): bỏ bớt dữ liệu vĩnh viễn để giảm kích thước (ảnh JPEG, nhạc MP3), không khôi phục được bản gốc — phù hợp với ảnh/âm thanh vì mắt/tai người khó nhận ra

phần bị bỏ. **Nén không mất dữ liệu (lossless)**, ví dụ **RLE**: thay một dãy giá trị lặp lại bằng cặp (giá trị, số lần lặp), khôi phục được **chính xác** bản gốc — bắt buộc dùng cho văn bản, chương trình, hay dữ liệu tài chính, vì chỉ cần sai 1 bit cũng có thể làm hỏng ý nghĩa hoặc khiến chương trình không chạy được.

1.8 Practice

Bài tập luyện tập

Convert `11001010` to denary and hexadecimal.

(A) 202 and `CA`
(C) 202 and `C9`
(B) 200 and `CA`
(D) 210 and `CA`

Lời giải chi tiết

Denary: $128+64+8+2 = 202$. Hex: nibbles `1100 1010` = `C A` = `CA`. Check: $CA_{16} = 12\times16+10 = 202$. **(A)**.

Bài tập luyện tập

Convert the denary value 233 to binary and hexadecimal (show your working).

Lời giải chi tiết

$233 = 128 + 64 + 32 + 8 + 1$, giving binary `11101001`. Splitting into nibbles: `1110 1001` = `E 9` = `E9`. Check: $E9_{16} = 14 \times 16 + 9 = 224 + 9 = 233$. □

Bài tập luyện tập

Convert the hexadecimal value `2F` to binary and denary.

(A) `00101111` and 47
(C) `00111111` and 63
(B) `00101110` and 46
(D) `00101111` and 45

Lời giải chi tiết

`2`=`0010`, `F`=`1111`, so binary is `00101111`. Denary: $2 \times 16 + 15 = 47$; check by place value $32 + 8 + 4 + 2 + 1 = 47$. **(A)**.

Bài tập luyện tập

Convert the 16-bit binary pattern `0010110111001001` to hexadecimal and denary.

Lời giải chi tiết

Split into nibbles: `0010 1101 1100 1001` = `2 D C 9` = `2DC9`. Denary from hex: $(2 \times 4096) + (13 \times 256) + (12 \times 16) + 9 = 8192 + 3328 + 192 + 9 = \boxed{11{,}721}$. Check by binary place value: $8192 + 2048 + 1024 + 256 + 128 + 64 + 8 + 1 = 11{,}721$. □

Bài tập luyện tập

Add the 8-bit binary numbers `01101011` (107) and `00111001` (57), showing the carry into each column, and state the denary result.

Lời giải chi tiết

Adding right to left with carries produces `10100100`. Column by column: $1+1=10$ (0, carry 1); $1+0+1=10$ (0, carry 1); $0+0+1=1$ (1, carry 0); $1+1=10$ (0, carry 1); $0+1+1=10$ (0, carry 1); $1+1+1=11$ (1, carry 1); $1+0+1=10$ (0, carry 1); $0+0+1=1$ (1, carry 0). Result `10100100` $=128+32+4=164$, matching $107+57=164$. No overflow, since 164 fits within the 8-bit unsigned range (0–255).

Bài tập luyện tập

An 8-bit unsigned register is used to add `11001000` (200) and `01100100` (100). What denary value is actually stored, and why does it not equal $200+100$?

(A) 44, because the carry out of the register is discarded (overflow)

(B) 300, the addition is always exact

(C) 0, because the register resets

(D) 156, due to rounding

Lời giải chi tiết

Adding the two patterns column by column gives `00101100` with a final carry-out of 1. Since the register only holds 8 bits, that carry-out is discarded, leaving `00101100`$=32+8+4=44$ stored. The true sum $200+100=300$ needed 9 bits ($100101100_2=300$), so it did not fit in the 8-bit register — this is **overflow**, and indeed $300-256=44$. **(A)**.

Bài tập luyện tập

Represent -100 in 8-bit two's complement.

(A) `10011100`

(B) `10011011`

(C) `01100100`

(D) `11100100`

Lời giải chi tiết

$100=$ `01100100`. Invert: `10011011`. Add 1: `10011100`. **(A)**. (Check: $-128+16+8+4=-100$. ✓)

Bài tập luyện tập

What denary value does the 8-bit two's complement pattern `11110001` represent?

(A) -15

(B) -17

(C) 241

(D) -113

Lời giải chi tiết

Invert `11110001` → `00001110`; add 1 → `00001111`$=15$. So the pattern represents -15. **(A)**. Check directly using negative place value for the leading bit ($-128, 64, 32, 16, 8, 4, 2, 1$): $-128+$

$64 + 32 + 16 + 1 = -15$. □

Bài tập luyện tập

What denary value does the 8-bit two's complement pattern `10001101` represent?

Lời giải chi tiết

Leading bit is 1 (negative). Invert: `01110010`; add 1: `01110011`$= 64 + 32 + 16 + 2 + 1 = 115$. So the value is -115. Check directly by negative place value: $-128 + 8 + 4 + 1 = -115$. □

Bài tập luyện tập

Add 60 and -25 in 8-bit two's complement using ordinary binary addition, and confirm there is no overflow.

Lời giải chi tiết

$60 =$ `00111100`. For -25: $25 =$ `00011001`, invert $\rightarrow$`11100110`, add 1 $\rightarrow$`11100111`.

```
   0 0 1 1 1 1 0 0   (60)
+  1 1 1 0 0 1 1 1   (-25)
-----------------------
1  0 0 1 0 0 0 1 1
```

Discarding the leading carry-out, the retained result is `00100011`$= 32 + 2 + 1 = 35$, matching $60 - 25 = 35$. Both operands have different sign bits (one positive, one negative), so by the overflow rule this addition **can never overflow** — and indeed the sign bit of the result (0) is correct.

Bài tập luyện tập

Add 100 and 90 in 8-bit two's complement. State the stored result and explain, using the overflow rule, why it is incorrect.

Lời giải chi tiết

$100 =$ `01100100`, $90 =$ `01011010`. Adding gives `10111110`, which (as two's complement: invert `01000001`, add 1 `01000010`$= 66$) reads as -66. Both operands are positive (sign bit 0), but the result's sign bit is 1 (negative) — by the overflow rule (same-sign operands, opposite-sign result), **overflow has occurred**. The true sum $100 + 90 = 190$ exceeds the maximum positive 8-bit two's complement value (127), so it wraps: $190 - 256 = -66$, matching the incorrect stored value.

Bài tập luyện tập

Add -90 and -70 in 8-bit two's complement. State the stored result and explain whether overflow has occurred.

(A) 96; overflow, since two negatives produced a positive result

(B) -160; no overflow

(C) 96; no overflow, since the addition is always exact

(D) -96; overflow

Lời giải chi tiết

-90 = 10100110, -70 = 10111010. Adding (and discarding the leading carry-out) gives 01100000= $64 + 32 = 96$. Both operands are negative (sign bit 1), but the result's sign bit is 0 (positive) — this satisfies the overflow rule, so **overflow has occurred**. The true sum is $-90 + (-70) = -160$, more negative than the 8-bit minimum of -128, so it wraps: $-160 + 256 = 96$, matching the stored (incorrect) value. **(A)**.

Bài tập luyện tập

State, in general terms, the rule used to detect overflow when adding two 8-bit two's complement numbers, and explain why adding a positive number to a negative number can never trigger it.

Lời giải chi tiết

Rule: overflow has occurred if and only if the two numbers added have the *same* sign bit, but the result has the *opposite* sign bit. When one operand is positive and the other negative, the true mathematical result must lie somewhere *between* the two operands in value — it can never be more positive than the larger positive-tending operand nor more negative than the more-negative operand, so it always stays safely within the representable range. Overflow can therefore only happen when *both* operands push the result in the *same* direction (both positive, or both negative).

Bài tập luyện tập

How many unique character codes are available in 8-bit extended ASCII, and how does this compare with standard 7-bit ASCII?

(A) 256 codes, versus 128 for standard ASCII
(B) 128 codes, versus 256 for standard ASCII
(C) 256 codes, versus 256 for standard ASCII
(D) 64 codes, versus 128 for standard ASCII

Lời giải chi tiết

Extended ASCII uses all 8 bits of a byte, giving $2^8 = 256$ possible codes, compared with $2^7 = 128$ codes for standard 7-bit ASCII — exactly double, since one extra bit doubles the number of possible patterns. **(A)**.

Bài tập luyện tập

Two computers exchange a text file, but one uses a different **code page** from the other for the byte values above 127. Explain what problem this can cause, and why Unicode avoids it.

Lời giải chi tiết

Extended ASCII's top 128 codes (128–255) were never universally standardised; different code pages map these same byte values to *different* characters (e.g. accented letters or symbols). If the receiving computer interprets a byte using the wrong code page, it will display the **wrong character** for that code, corrupting any part of the text that used codes above 127. Unicode avoids this because it defines one single, universally agreed code point for every character across virtually all languages and symbols, so there is no competing "code page" to cause ambiguity — the same byte sequence (interpreted with the same Unicode encoding,

e.g. UTF-8) means the same character everywhere.

Bài tập luyện tập

Explain why a multilingual messaging app (supporting English, Vietnamese, Chinese and emoji) must use Unicode rather than standard ASCII.

Lời giải chi tiết

Standard ASCII only provides $2^7 = 128$ character codes, which is barely enough for unaccented English letters, digits and punctuation — it has no codes for Vietnamese diacritics, Chinese characters, or emoji. Unicode uses many more bits per character, giving well over a million possible codes, which is large enough to include characters from essentially every world language plus symbols and emoji in a single, consistent character set.

Bài tập luyện tập

An uncompressed bitmap is 200 pixels $\times$ 100 pixels with a colour depth of 8 bits. What is the file size in KB (1 KB $= 1024$ bytes)?

(A) 19.53 KB
(C) 160 KB
(B) 20 KB
(D) 2.5 KB

Lời giải chi tiết

Bits $= 200 \times 100 \times 8 = 160{,}000$. Bytes $= 160{,}000 \div 8 = 20{,}000$. KB $= 20{,}000 \div 1024 \approx 19.53$ KB. **(A)**.

Bài tập luyện tập

An uncompressed bitmap is 640×480 pixels with a colour depth of 8 bits. Find its file size in KB, then find the new file size if the colour depth is increased to 24 bits.

Lời giải chi tiết

At 8 bits: $640 \times 480 \times 8 = 2{,}457{,}600$ bits $= 307{,}200$ bytes $= 307{,}200 \div 1024 = 300$ KB exactly. Increasing colour depth to 24 bits multiplies the size by $24 \div 8 = 3$ (resolution unchanged), giving $300 \times 3 = \boxed{900 \text{ KB}}$. Check directly: $640 \times 480 \times 24 = 7{,}372{,}800$ bits $= 921{,}600$ bytes $= 900$ KB. □

Bài tập luyện tập

Explain why an image file's header must store metadata such as width, height and colour depth, and what could go wrong if this metadata were missing or corrupted.

Lời giải chi tiết

The pixel data in an image file is just a long stream of bits; software needs to know how to **divide up** that stream — how many bits belong to each pixel (the colour depth) and how many pixels make up each row (the width, used together with the height to know the total number of pixels/rows). This information is stored as metadata in the file's header. If it were missing or corrupted, the software could not correctly split the stream into pixels and rows: it might

misinterpret the wrong number of bits as one pixel's colour, or wrap the pixel stream to the wrong row width, producing a distorted, discoloured, or completely unreadable image even though the underlying pixel data itself was never damaged.

Bài tập luyện tập

A monochrome bitmap row contains 40 white pixels, then 15 black pixels, then 25 white pixels. (a) Write the run-length encoded representation as (value, count) pairs. (b) Explain why RLE compresses this row well but would compress a typical colour photograph poorly.

Lời giải chi tiết

(a) `(W,40)(B,15)(W,25)` — 3 pairs instead of 80 individual pixel values.
(b) RLE saves space only when there are **long runs of identical values**. This bitmap row has exactly that (large blocks of a single colour). A photograph typically has **gradual, near-continuous colour variation** between neighbouring pixels, so there are few or no long runs of identical pixel values — RLE would produce little or no size reduction, and could even make the file larger (since each "run" of length 1 still needs a value and a count stored).

Bài tập luyện tập

A monochrome bitmap row has been run-length encoded as `(W,12)(B,3)(W,7)(B,18)`. How many pixels does the original row contain, and what is the 4th pixel's colour?

(A) 40 pixels; white
(B) 40 pixels; black
(C) 38 pixels; white
(D) 42 pixels; black

Lời giải chi tiết

Total pixels $= 12 + 3 + 7 + 18 = 40$. The first 12 pixels are white (positions 1–12), so the 4th pixel is **white**. **(A)**.

Bài tập luyện tập

Explain *why* lossy compression is appropriate for a photograph or a song, but unsuitable for compressing a spreadsheet of financial data or a program's executable file.

Lời giải chi tiết

A photograph or song contains fine detail that the human eye or ear cannot perceive very precisely; lossy compression discards exactly this imperceptible detail, so the loss of exact data is not noticed, while file size drops substantially. A spreadsheet of financial figures or an executable program, however, must be reproduced **exactly**: changing even a single bit could silently alter a number in an account, or corrupt an instruction so the program crashes or behaves incorrectly. Since lossy compression cannot guarantee an exact original on decompression, only **lossless** compression (e.g. ZIP-style algorithms or RLE) is acceptable for this kind of data, even though it typically achieves a smaller size reduction than a lossy method would.

Bài tập luyện tập

A mono audio clip is recorded at $48{,}000$ Hz, sample resolution 16 bits, for 5 seconds. It is then re-recorded in **stereo** using the same sample rate and resolution. Find the file size in KB for each version.

Lời giải chi tiết

Mono: bits $= 48{,}000 \times 16 \times 5 = 3{,}840{,}000$ bits $= 480{,}000$ bytes $= 480{,}000 \div 1024 \approx 468.75$ KB. **Stereo:** doubles the samples stored (one set per channel): bits $= 48{,}000 \times 16 \times 5 \times 2 = 7{,}680{,}000$ bits $= 960{,}000$ bytes $= 960{,}000 \div 1024 = \boxed{937.5 \text{ KB}}$ — exactly double the mono size, since stereo stores a second, independent channel of samples.

Bài tập luyện tập

Explain, in terms of file size and sound quality, the trade-off involved in choosing a higher sample rate and higher sample resolution when recording audio.

Lời giải chi tiết

A higher **sample rate** takes more snapshots of the analogue wave per second, capturing rapid changes in the sound more accurately; a higher **sample resolution** stores each snapshot with more possible amplitude levels, capturing quieter details and reducing rounding error in each sample. Both improvements make the digital recording a closer match to the original analogue sound (higher playback quality) — but both also directly increase the number of bits that must be stored per second of audio, so file size rises in direct proportion to each. Recording quality and file size are therefore a direct trade-off: better quality always costs more storage (or transmission bandwidth).

Bài tập luyện tập

Convert 3.5 MB into kilobytes and into bytes, using the binary convention (1 KB $= 1024$ bytes, 1 MB $= 1024$ KB).

Lời giải chi tiết

KB: $3.5 \times 1024 = 3584$ KB. Bytes: $3584 \times 1024 = \boxed{3{,}670{,}016}$ bytes (equivalently $3.5 \times 1024 \times 1024 = 3.5 \times 1{,}048{,}576 = 3{,}670{,}016$ bytes).

Bài tập luyện tập

A hard drive is advertised as "500 GB" by its manufacturer, using the **decimal** convention (1 GB $= 10^9$ bytes). When plugged into a computer that reports capacity using the **binary** convention (1 GB $= 2^{30}$ bytes), roughly what capacity will the computer display, and why does it not show exactly 500 GB?

(A) About 465.7 GB; the same number of bytes is now divided by a larger binary GB

(B) Exactly 500 GB; the conventions do not matter

(C) About 537 GB; binary GB is smaller than decimal GB

(D) The drive will not be recognised

Lời giải chi tiết

The drive genuinely stores $500 \times 10^9 = 500{,}000{,}000{,}000$ bytes. Measured using the binary convention, $500{,}000{,}000{,}000 \div 2^{30} = 500{,}000{,}000{,}000 \div 1{,}073{,}741{,}824 \approx 465.7$ GB. The physical number of bytes has not changed — only the size of the "GB" unit used to describe it has, since a binary GB (2^{30} bytes) is larger than a decimal GB (10^9 bytes), so dividing the same byte count by a larger unit gives a smaller number. **(A)**.

Bài tập luyện tập

A monochrome bitmap row contains 60 black pixels, then 10 white pixels, then 30 black pixels. Write its run-length encoded representation, and state how many individual values must be stored compared with the original row.

Lời giải chi tiết

Encoded: `(B,60)(W,10)(B,30)`. This stores 3 pairs, i.e. 6 individual values (3 colours and 3 counts), compared with the 100 individual pixel values in the original uncompressed row — a substantial, fully reversible (lossless) reduction because the row consists of long runs of identical colours.

Bài tập luyện tập

A student writes a program that will store both large photographs (as JPEG) and its own executable code. Justify, for *each* of these two file types, whether lossy or lossless compression should be used.

Lời giải chi tiết

Photographs (JPEG): lossy compression is appropriate, since photographs contain fine colour detail that the human eye cannot fully perceive; discarding the least noticeable detail gives a much smaller file with no perceptible loss of quality for most viewing purposes.
Executable program code: lossless compression must be used, since every single bit of a program's machine code instructions is meaningful — losing or altering even one bit could corrupt an instruction, causing the program to crash, behave unpredictably, or fail to run at all. Only a compression method that guarantees an exact, bit-for-bit original on decompression is acceptable here.

Bài tập luyện tập

A leaderboard app needs to store any whole number score from 0 to 150 inclusive. What is the smallest number of bits needed to store one score?

(A) 8 bits
(B) 7 bits
(C) 6 bits
(D) 150 bits

Lời giải chi tiết

The range 0 to 150 inclusive contains $150 - 0 + 1 = 151$ distinct values. $2^7 = 128$, which is less than 151 (not enough patterns). $2^8 = 256$, which is ≥ 151. So the smallest suitable width is **8 bits**. **(A)**.

Bài tập luyện tập

A scanner stores a 1000×800 pixel document using 8-bit greyscale. (a) Find the file size in KB. (b) State how many times larger the file would be if stored instead in 24-bit true colour, and explain why in terms of RGB channels.

Lời giải chi tiết

(a) Bits $= 1000 \times 800 \times 8 = 6{,}400{,}000$; bytes $= 6{,}400{,}000 \div 8 = 800{,}000$; KB $= 800{,}000 \div 1024 \approx 781.25$ KB.
(b) 24-bit true colour uses $24 \div 8 = 3$ times as many bits per pixel as 8-bit greyscale, so the file would be exactly **3 times larger** (≈ 2343.75 KB). This is because 24-bit colour is composed of three separate 8-bit channels — Red, Green and Blue — each stored independently for every pixel, whereas 8-bit greyscale stores only a single intensity channel per pixel.

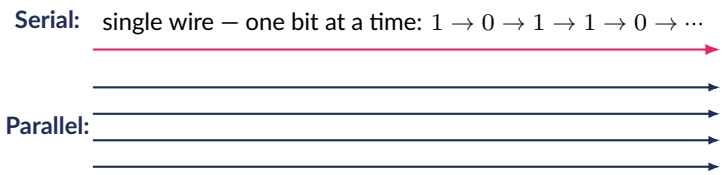
CAMBRIDGE IGCSE • Unit 2

Data Transmission

Mục tiêu Unit: Truyền dữ liệu nối tiếp/song song (bao gồm hiện tượng lệch tín hiệu và nhiễu xuyên âm), chuẩn USB, các chế độ truyền (simplex/half-duplex/full-duplex), các phương pháp phát hiện và định vị lỗi (parity một chiều, parity hai chiều, checksum, echo check, check digit, ARQ), và mã hoá dữ liệu (đối xứng, mã Caesar, và mã hoá bất đối xứng/khoá công khai).

2.1 Serial and parallel data transmission

Serial transmission sends bits one at a time, one after another, along a single wire (or a single twisted wire pair). **Parallel transmission** sends several bits simultaneously, side by side, along several separate wires bundled together — for example, 8 wires can carry all 8 bits of a byte in a single time step. At first glance parallel looks strictly faster, since it moves many bits per clock cycle instead of one; in practice, however, its reliability collapses over distance, which is why almost every external and long-distance connection used in modern computing (USB, SATA, Ethernet, Wi-Fi) is serial, while parallel transmission survives only inside very short, carefully engineered paths such as an internal CPU-to-memory bus.



Serial: 1 bit/wire/time step (robust over distance). Parallel: many bits/time step (fast but only short-distance safe, due to skew and crosstalk).

VN **GIẢI THÍCH TIẾNG VIỆT**

Truyền nối tiếp (serial): gửi từng bit một trên một dây — ổn định ở khoảng cách xa, dùng cho USB, SATA, mạng Ethernet. **Truyền song song (parallel)**: gửi nhiều bit cùng lúc trên nhiều dây — nhanh hơn về lý thuyết nhưng chỉ đáng tin cậy ở khoảng cách rất ngắn (ví dụ bus nội bộ trong máy tính) vì hiện tượng **lệch tín hiệu (skew)** và **nhiễu xuyên âm (crosstalk)**.

2.1.1 Skew and crosstalk

Two physical effects explain why parallel transmission becomes unreliable as cable length and clock speed increase.

Skew

Skew occurs because no two wires in a parallel cable are perfectly identical: tiny differences in wire length, resistance, and manufacturing mean an electrical signal travels along each wire at a very slightly different speed. When several bits are launched *at the same instant* on different wires, they no longer arrive at the receiver at exactly the same instant. If the receiver samples

all the wires together at a single moment (as it must, to reconstruct one byte), some bits may not have arrived yet while others already have — so the reconstructed byte can be wrong even though every individual bit travelled correctly along its own wire. Skew gets *worse* as the cable gets longer (small per-wire timing differences accumulate over distance) and as the clock speed increases (the safety margin between valid sampling windows shrinks).

Crosstalk

Crosstalk is electromagnetic interference between adjacent wires that are bundled tightly together. A changing electrical signal on one wire generates a small, fluctuating magnetic field; if another data wire runs close beside it, that field induces an unwanted, unintended voltage in the neighbouring wire, corrupting the signal it is meant to be carrying. The more wires bundled into a parallel cable, and the higher the switching frequency, the worse crosstalk becomes.

VN **GIẢI THÍCH TIẾNG VIỆT**

Lệch tín hiệu (skew): các dây trong cáp song song không hoàn toàn giống nhau về chiều dài/điện trở, nên các bit gửi *cùng lúc* lại đến đích ở các thời điểm hơi khác nhau — khiến bộ nhận ghép sai các bit thành một byte. **Nhiễu xuyên âm (crosstalk)**: tín hiệu điện trên một dây tạo ra từ trường nhỏ, gây nhiễu lên dây bên cạnh nằm sát nó. Cả hai hiện tượng đều **tệ hơn khi cáp dài hơn** hoặc **tốc độ xung nhịp cao hơn**.

Reducing crosstalk in practice: cables are often built from **twisted pairs** — two wires twisted around each other along their whole length, each carrying the signal and its exact inverse (**differential signalling**). Any outside interference tends to affect both wires in the pair almost identically; since the receiver reads the *difference* between the two wires rather than either wire alone, interference that affects both wires equally cancels out. This is why Ethernet network cables and many serial standards are built from twisted-pair wiring rather than plain parallel wires laid side by side.

VN **GIẢI THÍCH TIẾNG VIỆT**

Để giảm crosstalk, dây tín hiệu thường được **xoắn thành từng cặp (twisted pair)** và truyền tín hiệu cùng với **tín hiệu đảo ngược** của nó (**truyền vi sai**). Vì nhiễu tác động gần như giống nhau lên cả hai dây trong cặp, bộ nhận chỉ cần lấy **hiệu số** giữa hai dây để loại bỏ phần nhiễu chung — đây là lý do cáp mạng Ethernet và nhiều chuẩn nối tiếp hiện đại dùng dây xoắn đôi.

2.1.2 Choosing between serial and parallel: typical uses

Because skew and crosstalk scale with distance, engineers restrict parallel transmission to situations where the wires can be kept short, of equal length, and well-isolated: the internal buses connecting the CPU to main memory on a single circuit board, or (historically) very short ribbon cables such as old Parallel ATA (PATA) hard drive cables and Centronics printer cables, which were limited to under half a metre for reliable operation. Serial transmission avoids skew entirely (there is only one data path, so there is nothing for separate bits to get out of step with) and is far less exposed to crosstalk, which is why it is used for essentially all connections that must travel more than a few centimetres: USB, SATA, HDMI, Ethernet cabling, and all long-distance network links. Modern serial standards also run at extremely high clock speeds, so in practice serial links now routinely out-perform the parallel designs they replaced.

Rule of thumb: the greater the physical distance a signal must travel, the stronger the case for serial transmission. Parallel transmission is reserved for very short, tightly-controlled internal connections where the timing of every wire can be engineered to match closely.

2.1.3 Universal Serial Bus (USB)

USB is a serial standard that is **plug-and-play** (the operating system automatically detects and configures the device without manual setup), **hot-swappable** (devices can be connected or disconnected without restarting the computer), can supply **power** to the connected device over the same cable, and uses a **standardised connector**, replacing the many different, mutually incompatible ports that older peripherals required.

Ví dụ mẫu · Worked example

A student is giving a presentation. Partway through, she plugs a USB flash drive into her laptop (which is running and projecting to a screen), copies a backup file across, unplugs the drive, and immediately plugs the *same* drive into a classmate's laptop to continue working — without restarting either machine or installing any driver software. Explain which properties of USB make this possible.

Hot-swappable: the flash drive can be plugged in and removed while both laptops remain switched on and running, with no restart required. **Plug-and-play**: each laptop's operating system automatically recognises the drive and makes it available immediately, with no manual driver installation needed. **Standardised connector**: the same USB connector fits both laptops' USB ports, regardless of the make or model of computer.

VN **GIẢI THÍCH TIẾNG VIỆT**

USB là chuẩn nối tiếp có tính **plug-and-play** (hệ điều hành tự nhận diện thiết bị), **hot-swappable** (cắm/rút mà không cần khởi động lại máy), có thể **cấp nguồn** cho thiết bị, và dùng **đầu nối tiêu chuẩn hoá** chung cho nhiều loại thiết bị khác nhau.

2.1.4 Simplex, half-duplex and full-duplex

Transmission can also be classified by the *direction(s)* data is allowed to flow.

Khái niệm cốt lõi

Simplex: data flows in *one direction only*, never the reverse. **Half-duplex**: data can flow in *both* directions, but only *one direction at a time* — the two parties must take turns. **Full-duplex**: data flows in *both directions simultaneously*, with no need to take turns.

Mode	Definition	Real-world analogy
Simplex	Data travels one way only	Keyboard → computer (the keyboard never receives data back)
Half-duplex	Both directions possible, but only one at a time	Walkie-talkie (you must release the button to let the other person speak)
Full-duplex	Both directions at the same time	Telephone call (both people can talk and listen simultaneously)

Comparing simplex, half-duplex and full-duplex transmission with everyday analogies.

VN **GIẢI THÍCH TIẾNG VIỆT**

Simplex: dữ liệu chỉ truyền một chiều (vd bàn phím → máy tính). **Half-duplex**: truyền được hai chiều nhưng **chỉ một chiều tại một thời điểm** (vd bộ đàm — phải buông nút mới nghe được người kia nói). **Full-duplex**: truyền hai chiều **đồng thời** (vd cuộc gọi điện thoại — hai người có thể nói và nghe cùng lúc).

2.2 Error detection and correction

Data travelling along any transmission medium can be corrupted by electrical noise, interference, or hardware faults, so the receiver needs a reliable way to check whether the data it received matches what was actually sent. The methods in this section trade off differently between *how much extra data* they require, *whether* they can only detect an error or also locate/correct it, and *how* an error, once found, actually gets fixed — some methods (two-dimensional parity) can repair a single-bit error using only information already transmitted, while others (checksum, simple parity, echo check) can only flag that something is wrong and must rely on a separate mechanism, such as ARQ, to obtain a corrected copy of the data.

2.2.1 Parity check

Khái niệm cốt lõi

An extra **parity bit** is added to each byte so that the total count of 1-bits (data bits plus the parity bit) is always **even** (**even parity**) or always **odd** (**odd parity**), by agreement between sender and receiver. On arrival, the receiver recounts the 1-bits in the byte; if the count no longer matches the agreed parity, an error is flagged. A simple (one-dimensional) parity check has two important limitations: it **cannot detect an even number of bit-flips** within the same byte (because flipping an even number of bits leaves the total count of 1s with the same odd/even property it started with), and even when it does detect an error, it **cannot say which particular bit is wrong** — only that *some* bit in the byte is incorrect.

Ví dụ mẫu · Worked example

A system uses **even parity**. The 7 data bits are `1011001`. What is the parity bit, and what is the full 8-bit byte transmitted?

Count the 1s in `1011001`: there are 4 (already even), so the parity bit must be `0` to keep the total even. Byte transmitted: `10110010`.

Ví dụ mẫu · Worked example

A system using **even parity** receives the byte `11010101`. Has an error occurred?

Count the 1s: $1+1+0+1+0+1+0+1=5$ (odd). Since the system expects an **even** count of 1s, this mismatch means a **transmission error has been detected**.

Ví dụ mẫu · Blind spot of simple parity

A system uses **even parity**. The 7 data bits `1010011` are sent (4 ones, already even, so parity bit `0`; transmitted byte `10100110`). During transmission, *two* data bits are corrupted: bit 1 flips $1 \to 0$ and bit 3 flips $1 \to 0$. Show that the parity check does *not* detect this error.

Original data: `1010011`. After both flips: `0000011` (position 1 and position 3 changed; the parity bit itself is unaffected and still arrives as `0`). Received byte: `00000110`. Count of 1s $= 0+0+0+0+0+1+1+0 = 2$, which is still **even** — the check *passes*, even though two bits are wrong. This confirms that simple parity cannot detect an even number of bit errors within one byte.

VN **GIẢI THÍCH TIẾNG VIỆT**

Parity: thêm 1 bit để tổng số bit 1 luôn chẵn (even parity) hoặc luôn lẻ (odd parity); nếu số lượng bit 1 nhận được không khớp quy ước, lỗi bị phát hiện. Tuy nhiên parity **một chiều** có hai điểm yếu: không phát hiện được nếu có **số lượng chẵn bit bị lỗi** (vd đúng 2 bit cùng lật), và dù phát hiện được lỗi cũng **không biết chính xác bit nào sai**.

2.2.2 Two-dimensional parity (parity blocks)

Khái niệm cốt lõi

Two-dimensional parity overcomes the "cannot locate the error" weakness of simple parity by arranging a block of data bytes into a grid, and adding parity bits in *two* directions: a **row parity bit** at the end of every byte (row), *and* a **column parity bit** underneath every bit-column, calculated down that column across all the bytes in the block. If exactly one bit in the grid is flipped during transmission, recalculating parities will show *exactly one row* failing its check *and exactly one column* failing its check — and the corrupted bit must lie at the **intersection of that row and that column**. This lets the receiver not only **detect** that an error occurred, but **locate** precisely which bit is wrong; since flipping a single bit back to its correct value restores the whole block, the receiver can also **correct** a single-bit error without needing retransmission.

Ví dụ mẫu · Worked example

A block of 4 bytes uses **even parity** in both directions. The correctly-transmitted grid (row parity in the rightmost column, column parity in the bottom row, even parity throughout) is:

Data bits								Row parity
1	1	0	0	1	0	1	0	0
0	1	1	0	0	1	1	0	0
1	0	0	1	1	1	0	0	0
0	0	1	1	0	0	1	1	0
0	0	0	0	0	0	1	1	0

(bottom row: column parity bits; bottom-right corner: parity of the parity bits)

Suppose the bit in **row 3, column 5** is flipped during transmission ($1 \rightarrow 0$), so row 3 is received as `10010100`. Show how the receiver locates the error.

Row check: recompute the count of 1s in each received row. Row 1: 4 (even, OK). Row 2: 4 (even, OK). Row 3 (received `1 0 0 1 0 1 0 0`): $1+0+0+1+0+1+0+0 = 3$, which is **odd** — but the stored row parity is `0` (even expected) $\Rightarrow$ **Row 3 fails**. Row 4: 4 (even, OK).

Column check: recompute the count of 1s down each column using the received data. Columns 1,2,3,4,6,7,8 all still sum to an even number and match their stored column parity. Column 5 (data down the column: row1$=1$, row2$=0$, row3$=0$ [received, flipped], row4$=0$): sum $=1$, which is **odd** — but the stored column parity is `0` (even expected) $\Rightarrow$ **Column 5 fails**.

Conclusion: the only row that fails is Row 3, and the only column that fails is Column 5, so the corrupted bit is at their intersection — **row 3, column 5**, exactly where the flip occurred. Since only one bit is wrong, the receiver corrects it by flipping that bit back ($0 \to 1$), fully repairing the block without needing a retransmission.

VN **GIẢI THÍCH TIẾNG VIỆT**

Parity hai chiều (two-dimensional parity): sắp dữ liệu thành lưới, thêm **parity bit theo hàng** và **parity bit theo cột**. Nếu đúng 1 bit bị lỗi, sẽ có đúng **1 hàng** và **1 cột** báo sai — bit lỗi nằm tại **giao điểm** của hàng và cột đó. Nhờ vậy bộ nhận không chỉ **phát hiện** mà còn **định vị và sửa** được lỗi 1 bit mà không cần yêu cầu gửi lại.

2.2.3 Checksum

Khái niệm cốt lõi

The sender calculates a single value (the **checksum**) from an *entire block* of data using an agreed formula (for example, summing the denary value of every byte in the block, then taking the result modulo some fixed number so the checksum itself fits in one byte), and appends this value to the end of the block. On arrival, the receiver performs *exactly the same calculation* on the data it received and compares its own result to the transmitted checksum. If the two values match, the block is (very likely) intact; if they differ, at least one error occurred somewhere in the block.

Ví dụ mẫu · Worked example

A block of four bytes with denary values $58, 231, 12, 145$ is transmitted, along with a checksum calculated as (sum of all bytes) modulo 256.

Sum $= 58 + 231 + 12 + 145 = 446$. Checksum $= 446 \bmod 256 = 446 - 256 = \boxed{190}$; this value (190) is appended to the block and transmitted.

Suppose that, due to a transmission fault, the second byte (231) is corrupted and arrives as 227. The receiver recalculates: $58 + 227 + 12 + 145 = 442$; $442 \bmod 256 = 442 - 256 = 186$. Since the recalculated value (186) does *not* match the transmitted checksum (190), the receiver concludes an error occurred somewhere in the block.

VN **GIẢI THÍCH TIẾNG VIỆT**

Checksum: tính một giá trị duy nhất từ **toàn bộ khối dữ liệu** (thường là tổng các byte, lấy modulo để giá trị vừa 1 byte), gửi kèm khối dữ liệu. Bên nhận tính lại giá trị này từ dữ liệu nhận được; nếu **không khớp** với checksum được gửi, kết luận có lỗi xảy ra đâu đó trong khối.

2.2.4 Echo check

Khái niệm cốt lõi

In an **echo check**, the receiver simply sends the data it received straight back to the original sender. The sender then compares this "echoed" copy to the data it originally transmitted; if the two do not match exactly, an error is signalled and the data can be resent. Unlike parity or checksum, an echo check requires *no* calculation at all — but because the *entire* block of data must be transmitted a second time (once forward, once back), it effectively **doubles the total**

network traffic needed to send the same amount of useful data, which makes it considerably less efficient for large volumes of data or slow/expensive links.

Ví dụ mẫu · Worked example

A sender transmits the text `DATA` to a receiver using an echo check. The receiver echoes back exactly what it received, and the sender compares the echo to the original.

No error case: sender sends `DATA` → receiver receives `DATA` and echoes `DATA` back → sender compares `DATA` (original) with `DATA` (echo) — they match, so no error is reported.

Error case: sender sends `DATA` → due to interference, the receiver actually receives `DATB` and echoes `DATB` back → sender compares `DATA` (original) with `DATB` (echo) — they do *not* match, so the sender knows an error occurred somewhere along the way and can request the data again.

Notice that in both cases, the four characters of data were transmitted *twice* in total (once each direction) merely to check *one* short message — illustrating why echo check roughly doubles network traffic compared to checksum, which sends only a small extra value alongside the data.

VN **GIẢI THÍCH TIẾNG VIỆT**

Echo check: bên nhận gửi **nguyên văn dữ liệu nhận được** trả lại cho bên gửi; bên gửi so sánh với dữ liệu gốc, nếu khác nhau thì có lỗi. Không cần tính toán như parity/checksum, nhưng **gấp đôi lưu lượng mạng** vì dữ liệu phải truyền đi rồi truyền lại.

2.2.5 Check digits

Khái niệm cốt lõi

A **check digit** is one extra digit added to the end of a numeric code (such as an **ISBN** book number or a retail **barcode**) and calculated from all the other digits using an agreed formula. Its purpose is to **validate that the code was typed or scanned correctly**: after entry, the same formula is applied to the digits actually received, and if the result does not match, the system knows immediately that a mistake was made (a mistyped digit, a misread barcode, two swapped digits, and so on), without needing any external reference to check against.

Ví dụ mẫu · Worked example

A 10-digit code uses a **modulo-11** check digit, calculated as follows: multiply each of the first 9 digits by a descending weight from 10 down to 2, sum the products, and find the remainder when this sum is divided by 11. The check digit is $11 -$ remainder (using the symbol `X` to represent the value 10, and using `0` if the remainder itself is 0).

For the first 9 digits `1 2 3 4 5 6 7 8 9`:

Digit	1	2	3	4	5	6	7	8	9
Weight	10	9	8	7	6	5	4	3	2
Product	10	18	24	28	30	30	28	24	18

Sum of products $= 10 + 18 + 24 + 28 + 30 + 30 + 28 + 24 + 18 = 210$. $210 \div 11 = 19$ remainder 1. Check digit $= 11 - 1 = 10 =$ `X`.

Full code: `123456789X`.

Verification: including the check digit with weight 1, the full weighted total is $210 + (10 \times 1) = 220$, and $220 \div 11 = 20$ exactly (remainder 0) — confirming the check digit `X` is correct.

Ví dụ mẫu · Detecting a mistyped digit

Suppose the code `123456789X` above is entered incorrectly as `133456789X` (the second digit, 2, mistyped as 3). Show that the check-digit calculation flags this as invalid.

Recompute the full weighted total (weights 10 down to 1, with `X`$= 10$): digits $1, 3, 3, 4, 5, 6, 7, 8, 9, 10$ with weights $10, 9, 8, 7, 6, 5, 4, 3, 2, 1$. Products: $10, 27, 24, 28, 30, 30, 28, 24, 18, 10$. Sum $= 10 + 27 + 24 + 28 + 30 + 30 + 28 + 24 + 18 + 10 = 229$. $229 \div 11 = 20$ remainder 9. Since the remainder is *not* 0, the code is **invalid** — the check-digit system has correctly detected that a digit was mistyped.

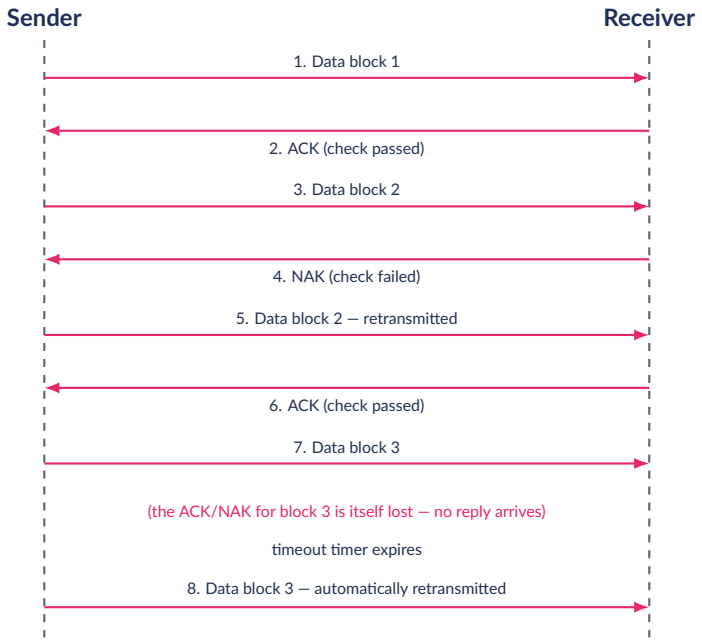
VN **GIẢI THÍCH TIẾNG VIỆT**

Check digit: một chữ số thêm vào cuối mã số (ISBN, mã vạch) được tính từ các chữ số còn lại bằng một công thức quy ước (ví dụ nhân theo trọng số giảm dần rồi lấy modulo 11). Khi nhập/quét lại mã, hệ thống tính lại và so sánh — nếu **không khớp**, phát hiện ngay có lỗi nhập liệu hoặc đọc mã sai, mà không cần tra cứu dữ liệu gốc ở đâu khác.

2.2.6 Automatic Repeat reQuest (ARQ)

Khái niệm cốt lõi

ARQ combines an error-detection method (parity or checksum) with an automatic retransmission protocol. After checking each block of data, the receiver returns an **ACK** (acknowledgement) if the block passed the check, or a **NAK** (negative acknowledgement) if it failed, requesting retransmission. As a safety net, the sender also starts a **timeout** timer whenever it transmits a block: if *no* ACK or NAK arrives before the timeout expires (for example, because the ACK/NAK itself was lost or corrupted in transit), the sender automatically retransmits the block anyway, without waiting to be asked.



ARQ timeline: blocks 1 and 2 are checked and acknowledged/rejected normally via ACK/NAK; block 3's acknowledgement is itself lost in transit, so the sender's own timeout forces an automatic retransmission.

VN **GIẢI THÍCH TIẾNG VIỆT**

ARQ: dùng **ACK** (đúng) / **NAK** (sai, yêu cầu gửi lại) sau khi kiểm tra parity/checksum. Nếu bên gửi **không nhận được phản hồi nào** trong thời gian **timeout** quy định (vd vì chính ACK/NAK bị mất), bên gửi vẫn **tự động gửi lại** dữ liệu mà không cần đợi yêu cầu, đây là lớp bảo vệ dự phòng.

Sequence numbers: because a timeout can cause the sender to retransmit a block that the receiver in fact already received correctly (if only the returning ACK was lost, not the original data), practical ARQ systems label each block with a **sequence number**. The receiver uses this number to spot and silently discard any **duplicate** block that arrives a second time, and to notice if a block appears to be **missing** from the expected sequence — something the ACK/-NAK/timeout mechanism alone cannot guarantee on its own.

VN **GIẢI THÍCH TIẾNG VIỆT**

Số thứ tự (sequence number): vì timeout có thể khiến bên gửi gửi lại một khối dữ liệu **đã đến nơi an toàn** (chỉ có ACK bị mất trên đường về), mỗi khối dữ liệu thường được đánh số thứ tự để bên nhận **phát hiện và bỏ qua bản sao trùng lặp**, đồng thời phát hiện nếu có khối nào bị thiếu trong chuỗi.

2.3 Encryption

2.3.1 Symmetric encryption

Encryption scrambles readable data (**plaintext**) into unreadable **ciphertext** using an algorithm and a **key**, so that data intercepted during transmission cannot be understood by anyone without the key. In **symmetric encryption**, the *same* key is used both to encrypt and to decrypt the data, so both sender and receiver must already possess (and keep completely secret) that identical key *before* secure communication begins. This creates the **key-distribution problem**: the key itself must somehow be shared safely first — if it is intercepted while being shared, everything encrypted with it can subsequently be read by the attacker.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mã hoá (encryption) biến dữ liệu gốc (**plaintext**) thành dữ liệu không đọc được (**ciphertext**) bằng một **khoá (key)**. Với **mã hoá đối xứng (symmetric)**, bên gửi và bên nhận dùng **cùng một khoá** để mã hoá/giải mã — rủi ro lớn nhất là **làm sao chia sẻ khoá đó một cách an toàn** trước khi liên lạc, vì nếu khoá bị lộ, toàn bộ dữ liệu mã hoá bằng khoá đó có thể bị đọc được.

2.3.2 The Caesar cipher: a simple worked example

Before studying real encryption algorithms, it helps to see the idea of "key + algorithm" concretely with the **Caesar cipher** (a simple **shift cipher**): every letter of the plaintext is shifted a fixed number of places along the alphabet, and that fixed number *is* the key. Decryption simply shifts every ciphertext letter back the same number of places.

Ví dụ mẫu · Worked example

Encrypt the plaintext `HELLO` using a Caesar cipher with key (shift) $= 3$.
Shift each letter 3 places forward in the alphabet: `H`→`K`, `E`→`H`, `L`→`O`, `L`→`O`, `O`→`R`.
Ciphertext: `KHOOR`.

Ví dụ mẫu · Worked example

Decrypt the ciphertext `KHOOR`, given that it was encrypted with a Caesar cipher using key $= 3$.
Shift each letter 3 places *back*: `K`→`H`, `H`→`E`, `O`→`L`, `O`→`L`, `R`→`O`.
Plaintext recovered: `HELLO` — confirming that decryption with the correct key exactly undoes the encryption.

(!) Lỗi thường gặp: The Caesar cipher is useful for building intuition, but it is **not secure**: there are only 25 possible non-trivial shift keys, so an attacker can simply try every shift in turn (a "brute-force" attack) or use **frequency analysis** (comparing how often each ciphertext letter appears against the known frequency of letters in ordinary English) to recover the plaintext within seconds, without ever knowing the key in advance. Real encryption algorithms use vastly larger keys (commonly 128 bits or more, giving far more possible keys than a brute-force search could ever try) precisely to avoid this weakness.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mã Caesar (Caesar cipher): dịch chuyển mỗi chữ cái đi một số vị trí cố định trong bảng chữ cái — số vị trí đó chính là **khoá**. Giải mã chỉ cần dịch ngược lại đúng số vị trí đó. Mã Caesar **không an toàn** trong thực tế vì chỉ có 25 khoá khả dĩ, dễ bị **dò thử toàn bộ** hoặc **phân tích tần suất chữ cái**.

2.3.3 Asymmetric (public-key) encryption

Khái niệm cốt lõi

In **asymmetric (public-key) encryption**, each user has a **key pair**: a **public key**, which is shared openly with anyone and used by *others* to encrypt messages intended for that user, and a **private key**, which is kept completely secret by its owner and is the *only* key that can decrypt messages that were encrypted with the matching public key. To send Bob a confidential message, Alice encrypts it using **Bob's public key** (which she can obtain openly, with no secrecy required); only Bob's **private key** — which he has never shared with anyone — can decrypt it. This completely avoids the key-distribution problem of symmetric encryption, because the key used to *encrypt* never needs to be kept secret at all, and the secret (private) key never has to travel anywhere or be shared with another party.

	Symmetric encryption	**Asymmetric (public-key) encryption**
Keys used	One single shared key, used for both encryption and decryption	Two different keys per user: a public key and a private key
Key secrecy	The one key must be kept secret by *both* sender and receiver	Only the private key is secret; the public key can be shared with anyone
Key-distribution problem	Present — the shared key must somehow be exchanged safely before communication	Solved — the public key needs no secure channel at all; the private key never leaves its owner
Typical speed	Faster, less computationally intensive	Slower, more computationally intensive

Comparing symmetric and asymmetric (public-key) encryption.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mã hoá bất đối xứng (asymmetric/public-key): mỗi người dùng có **cặp khoá — khoá công khai (public key)** chia sẻ thoải mái, dùng để mã hoá tin nhắn gửi *đến* người đó; **khoá riêng tư (private key)** giữ bí mật tuyệt đối, chỉ dùng để giải mã. Muốn gửi tin an toàn cho Bob, Alice mã hoá bằng **khoá công khai của Bob**; chỉ **khoá riêng của Bob** mới giải mã được. Cách này giải quyết được **vấn đề chia sẻ khoá** của mã hoá đối xứng vì khoá riêng không bao giờ phải gửi đi.

Bài tập luyện tập

A system uses **odd parity**. What parity bit should be appended to the data bits `1101100`?

(A) 1
(B) 0
(C) Cannot be determined
(D) Depends on the checksum

Lời giải chi tiết

Count the 1s in `1101100`: $1+1+0+1+1+0+0=4$ (even). Since odd parity requires the total to be **odd**, the parity bit must be 1 (making the total 5, odd). **(A)**.

Bài tập luyện tập

State *one* reason USB (a serial standard) is preferred over a parallel connection for connecting an external hard drive several metres away.

Lời giải chi tiết

Over longer cable runs, parallel transmission suffers from **skew** (bits sent at the same instant on different wires arrive at slightly different times because the wires are not perfectly identical) and **crosstalk** (electrical interference between adjacent wires), both of which corrupt data. Serial transmission uses a single data path, avoiding skew between wires and making it far more reliable at distance, which is why USB and most external connections are serial.

Bài tập luyện tập

A receiver recalculates a checksum for a received block of data and finds it does *not* match the checksum sent with the data. What should happen next under an ARQ system?

(A) The data is accepted anyway
(B) A NAK is sent and the sender retransmits the block
(C) The connection is permanently closed
(D) The parity bit is flipped to fix the error

Lời giải chi tiết

A checksum mismatch means an error was detected. Under ARQ, the receiver sends a **NAK** (negative acknowledgement), and the sender retransmits the affected block of data. **(B)**.

Bài tập luyện tập

Put the following steps of Automatic Repeat reQuest (ARQ) into the correct order: (i) receiver checks the data using parity/checksum, (ii) sender transmits a block of data, (iii) if no ACK/NAK is received within the timeout, sender retransmits, (iv) receiver sends ACK if correct or NAK if

incorrect.

Lời giải chi tiết

Correct order: **(ii)** sender transmits → **(i)** receiver checks the data → **(iv)** receiver sends ACK or NAK → **(iii)** if the sender gets no response within the timeout period, it retransmits automatically (as a safety net in case the ACK/NAK itself was lost).

Bài tập luyện tập

A company sends confidential salary data across the internet using symmetric encryption. Explain *one* practical risk with this approach and how it could be reduced.

Lời giải chi tiết

Risk: because the same key both encrypts and decrypts the data, that key must somehow be shared with the receiver *before* the secure communication starts. If this key is intercepted while being shared (e.g. sent by an insecure email), an attacker can decrypt every message protected by it. **Reduction:** share the key through a separate, secure channel (e.g. in person, or using a key-exchange protocol), change keys regularly, and never transmit the key alongside the encrypted data itself.

Bài tập luyện tập

A system uses **even parity**. What parity bit should be appended to the 7 data bits `1011000`, and what is the full 8-bit byte transmitted?

(A) `10110001`
(B) `10110000`
(C) `01011000`
(D) `11011000`

Lời giải chi tiết

Count the 1s in `1011000`: $1+0+1+1+0+0+0=3$ (odd). For **even** parity, the parity bit must be `1` to make the total even ($3+1=4$). Full byte: `10110001`. **(A)**.

Bài tập luyện tập

A system uses **even parity**. The 7 data bits `1010011` are transmitted as the byte `10100110`. During transmission, bit 1 and bit 3 (of the original 7 data bits) are both flipped. Does the parity check detect this error? Show your working and explain the general principle this illustrates.

Lời giải chi tiết

Original data `1010011` (4 ones, even, so parity bit `0`). After flipping bit 1 ($1 \to 0$) and bit 3 ($1 \to 0$): data becomes `0000011`. Parity bit is unaffected, so the received byte is `00000110`. Counting all 8 bits: $0+0+0+0+0+1+1+0=2$, which is **even** — the parity check **passes**, even though two bits are actually wrong. This illustrates that a simple (one-dimensional) parity check cannot detect an **even number** of bit errors within the same byte, because flipping an even number of bits does not change whether the total count of 1s is odd or even.

Bài tập luyện tập

A 3-byte block uses even parity in both directions (row and column). The block is received as shown below, where the stored row and column parity bits were calculated *before* transmission:

Received data								Row parity
1	1	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	0
1	0	1	1	0	1	1	0	1
0	0	0	0	1	1	0	1	1

Identify which single bit was corrupted during transmission, and state its correct original value.

Lời giải chi tiết

Row check (recount 1s in each received row, compare to stored row parity for even overall parity): Row 1: $1+1+1+0+0+0+1+0=4$ (even) matches parity 0 — OK. Row 2: $0+1+0+1+1+1+0+1=5$ (odd) but stored parity is 0 (even expected) — **Row 2 fails**. Row 3: $1+0+1+1+0+1+1+0=5$ (odd), stored parity is 1; total with parity $=5+1=6$ (even) — OK.

Column check (recount 1s down each column, compare to stored column parity): columns 1–5, 7, 8 all check out. Column 6 (values down the column: $0,1,1$) sums to 2 (even), but the stored column parity for column 6 is 1 (odd expected); total with parity $=2+1=3$ (odd) — **Column 6 fails**.

Since Row 2 and Column 6 are the only row and column to fail, the corrupted bit is at their intersection: **row 2, column 6**. It was received as 1 but the check shows it must be corrected back to 0 (so that row 2 becomes $0\ 1\ 0\ 1\ 1\ 0\ 0\ 1$, restoring even parity in both its row and column).

Bài tập luyện tập

A block of three bytes with denary values $76, 154, 201$ is transmitted with checksum $= 99$, using the rule checksum $=$ (sum of bytes) mod 256. (a) Verify this checksum is correct. (b) If the second byte is corrupted in transit and received as 99 instead of 154 (with the other two bytes and the checksum arriving unchanged), does the receiver detect an error?

Lời giải chi tiết

(a) Sum $= 76+154+201 = 431$. 431 mod $256 = 431-256 = 175$. This does *not* equal the stated checksum of 99 — so the originally stated checksum of 99 would actually be **incorrect**; the correct checksum for this block is $\boxed{175}$.

(b) Using the corrected checksum of 175: if byte 154 is corrupted to 99, the receiver recalculates $76+99+201=376$; 376 mod $256 = 376-256 = 120$. Since $120 \neq 175$ (the transmitted checksum), the receiver **does detect an error**.

Bài tập luyện tập

Which error-detection method requires the receiver to transmit the data it received directly back to the original sender for comparison, and as a result roughly doubles the network traffic needed?

(A) Parity check
(B) Checksum
(C) Echo check
(D) Automatic Repeat reQuest (ARQ)

Lời giải chi tiết

An **echo check** sends the received data straight back to the sender, which compares it with the original; because the data effectively travels the link twice (once each way), total network traffic roughly doubles compared to sending it only once. **(C)**.

Bài tập luyện tập

A 10-digit code uses a modulo-11 check digit: the first 9 digits are each multiplied by a descending weight from 10 to 2, the products summed, and the check digit is 11 minus the remainder when this sum is divided by 11 (using `X` for the value 10, and `0` if the remainder is already 0). The first 9 digits are `3 5 7 0 2 9 4 6 1`. (a) Calculate the check digit and state the full 10-character code. (b) Someone re-enters the code but mistypes the 4th digit (`0`) as `8`. Show that the check-digit calculation flags this as invalid.

Lời giải chi tiết

(a) Products: $3{\times}10 = 30$, $5{\times}9 = 45$, $7{\times}8 = 56$, $0{\times}7 = 0$, $2{\times}6 = 12$, $9{\times}5 = 45$, $4{\times}4 = 16$, $6{\times}3 = 18$, $1{\times}2 = 2$. Sum $= 30 + 45 + 56 + 0 + 12 + 45 + 16 + 18 + 2 = 224$. $224 \div 11 = 20$ remainder 4. Check digit $= 11 - 4 = 7$. Full code: `3570294617`.
Verification: total with check digit (weight 1) $= 224 + (7{\times}1) = 231 = 11 \times 21$, remainder 0 — confirms 7 is correct.

(b) With digit 4 changed from `0` to `8` (weight 7), the sum of the first 9 digits increases by $(8 - 0) \times 7 = 56$, giving $224 + 56 = 280$. Including the (unchanged) check digit: $280 + (7{\times}1) = 287$. $287 \div 11 = 26$ remainder 1 (since $11 \times 26 = 286$). Because the remainder is not 0, the code is correctly flagged as **invalid** — the mistyped digit has been detected.

Bài tập luyện tập

A sender transmits a block of data and starts a timeout timer, but receives no ACK or NAK before the timer expires. Explain what the sender does next, and why this behaviour is necessary even though the receiver might actually have received the data correctly.

Lời giải chi tiết

The sender **automatically retransmits** the block once the timeout expires, without waiting for any further instruction. This is necessary because the sender cannot distinguish between two very different situations: (i) the data itself was corrupted or never arrived, or (ii) the data arrived correctly and the receiver *did* send an ACK, but that acknowledgement was itself lost or corrupted on its way back. In case (ii), simply doing nothing would leave the sender waiting forever for a reply that will never come; retransmitting after a timeout guarantees the sender always eventually gets a successful exchange, even though it does occasionally cause the receiver to receive one block twice (which the receiver's protocol must also be designed to handle, e.g. using sequence numbers to spot and discard duplicates).

Bài tập luyện tập

Which statement correctly describes **crosstalk** in parallel data transmission?

(A) Electrical interference in which a changing signal on one wire induces an unwanted voltage in an adjacent wire

(B) Bits sent at the same instant on different wires arriving at slightly different times

(C) Loss of synchronisation between the sender's and receiver's clocks

(D) Permanent physical damage to a transmission wire

Lời giải chi tiết

Crosstalk is electromagnetic interference between wires bundled closely together: a changing signal on one wire generates a fluctuating field that induces an unwanted voltage on a neighbouring wire. **(A)**. (Option B describes *skew*, a separate effect.)

Bài tập luyện tập

Explain why the internal bus connecting a CPU to main memory can safely use parallel transmission, while an external hard drive connected by a cable several metres long should use a serial connection instead.

Lời giải chi tiết

The internal CPU-to-memory bus consists of very short wires (centimetres), manufactured to almost identical length and kept on a rigid, well-shielded circuit board, so the timing difference between wires (skew) and interference between wires (crosstalk) can both be kept extremely small and predictable. Over a cable several metres long, small per-wire differences in length/resistance accumulate into significant skew, and the wires are more exposed to external interference and mutual crosstalk, both of which corrupt data much more severely at that scale. A serial connection (a single data path) has nothing to be out of step with and is largely immune to skew, which is why external, longer-distance links such as hard drive cables use serial standards (e.g. SATA, USB) rather than parallel ones.

Bài tập luyện tập

Which of the following is an example of **full-duplex** transmission?

(A) A walkie-talkie conversation

(B) A telephone call, where both people can talk and listen at the same time

(C) Data sent from a keyboard to a computer

(D) A television broadcast signal

Lời giải chi tiết

Full-duplex means both directions of data flow happen **simultaneously**. A telephone call allows both parties to talk and listen at once. **(B)**. (A walkie-talkie is half-duplex; a keyboard-to-computer link and a TV broadcast are both simplex.)

Bài tập luyện tập

Explain why asymmetric (public-key) encryption avoids the key-distribution problem that affects symmetric encryption.

Lời giải chi tiết

In symmetric encryption, the *same* secret key is needed by both parties before communication starts, so that key itself must be transported or shared somehow — and if it is intercepted during that sharing, the encryption is completely broken. In asymmetric encryption, each user instead has a public key (which can be published or shared completely openly, since it is only used to *encrypt* messages *to* that user) and a private key (which is generated once and **never has to be shared or transmitted anywhere** — it stays with its owner at all times). Because the only key that ever needs to travel to anyone (the public key) does not need to be kept secret at all, there is no equivalent moment where a secret key is exposed to interception, so the key-distribution problem simply does not arise.

Bài tập luyện tập

The ciphertext `PDWK` was produced using a Caesar cipher with key (shift) $= 3$. Decrypt it, and state the plaintext.

(A) `MATH`
(B) `MATT`
(C) `PATH`
(D) `LATH`

Lời giải chi tiết

Shift each letter back 3 places: `P`→`M`, `D`→`A`, `W`→`T`, `K`→`H`. Plaintext: `MATH`. **(A)**.

Bài tập luyện tập

Explain *one* weakness of the Caesar cipher, and why modern encryption algorithms are designed with a much larger number of possible keys.

Lời giải chi tiết

The Caesar cipher has only 25 non-trivial shift keys (a shift of 0 would leave the message unchanged), so an attacker can simply try every possible shift in turn — a brute-force search that takes almost no time at all on a computer — or use **frequency analysis** (matching how often each ciphertext letter occurs against the known frequency of letters in normal English text) to recover the plaintext without even needing to guess the key directly. Modern encryption algorithms use keys with an enormous number of possible values (for example, a 128-bit key has 2^{128} possibilities), making a brute-force search computationally infeasible even for the most powerful computers, which is essential for keeping genuinely sensitive data secure.

Bài tập luyện tập

What is the main advantage of two-dimensional (row-and-column) parity over simple one-dimensional parity?

(A) It can both detect and locate (and, for a single-bit error, correct) the erroneous bit
(B) It doubles the available network bandwidth
(C) It removes the need for ACK/NAK acknowledgements entirely
(D) It encrypts the transmitted data as well as checking for errors

Lời giải chi tiết

Simple parity can only say *that* an error occurred somewhere in a byte, not which bit. Two-dimensional parity adds parity bits along both rows and columns of a data block, so a single-bit error causes exactly one row and one column to fail their checks — pinpointing the bit at their intersection, and allowing it to be corrected without retransmission. **(A)**.

Bài tập luyện tập

A byte protected only by simple (one-dimensional) even parity is received, and the parity check fails. (a) Can the receiver tell, from the parity check alone, *which* of the 8 bits is wrong? (b) What additional structure would need to be added to let the receiver both locate and correct a single-bit error?

Lời giải chi tiết

(a) No. A one-dimensional parity check only recomputes a single count (the total number of 1s in the byte) and compares it to the expected odd/even value; this tells the receiver only that *some* bit in the byte is wrong, not which position it occupies.
(b) The block of data would need to be arranged as a grid of multiple bytes, with an additional **column parity bit** calculated down each bit-position across all the bytes in the block (in addition to the existing row parity bit for each byte). With this two-dimensional parity structure, a single-bit error causes exactly one row and one column to fail, and the intersection of that row and column identifies (and, since only one bit is wrong, allows correction of) the erroneous bit.

Bài tập luyện tập

Alice wants to send Bob a confidential message using asymmetric (public-key) encryption. Which key should Alice use to encrypt the message?

(A) Alice's own private key
(B) Alice's own public key
(C) Bob's public key
(D) Bob's private key

Lời giải chi tiết

To send a confidential message *to* Bob, the sender encrypts using the **recipient's public key** (freely available), so that only the matching **private key** — held solely by Bob — can decrypt it. **(C)**.

Bài tập luyện tập

A block of three bytes with denary values $12, 250, 64$ is transmitted with checksum $=$ (sum of bytes) mod 256, under an ARQ protocol. (a) Calculate the checksum sent with the block. (b) During transmission, the second byte is corrupted from 250 to 254. Show what the receiver calculates, and describe what happens next under ARQ.

Lời giải chi tiết

(a) Sum $= 12 + 250 + 64 = 326$. 326 mod $256 = 326 - 256 = 70$. Checksum transmitted $= 70$.
(b) The receiver receives bytes $12, 254, 64$ (254 in place of 250) together with the transmitted checksum 70. It recalculates: $12 + 254 + 64 = 330$; 330 mod $256 = 330 - 256 = 74$. Since $74 \neq 70$, the receiver detects an error and sends a **NAK** to the sender. Under ARQ, the sender

then **retransmits** the block of three bytes; this repeats (with a timeout-triggered retransmission as backup, in case the NAK itself is lost) until a checksum eventually matches and an ACK is returned.

Bài tập luyện tập

What is the primary purpose of a **check digit** attached to a code such as an ISBN or retail barcode?

(A) To encrypt the numeric code so it cannot be read by unauthorised parties

(B) To detect errors made when the code is typed in or scanned, by recalculating a formula and comparing the result

(C) To compress the numeric code into fewer digits

(D) To authenticate the identity of the person entering the code

Lời giải chi tiết

A check digit is calculated from the other digits of a code using an agreed formula; recalculating that formula from the digits as entered or scanned, and comparing to the check digit, reveals whether the code was captured correctly. **(B)**. It plays no role in encryption, compression, or identity authentication.

Bài tập luyện tập

A simple modulo-11 check-digit scheme reliably detects a single mistyped digit. Explain why it can nonetheless fail to detect certain other types of transcription error, such as two compensating digit changes.

Lời giải chi tiết

The check-digit calculation only tests whether the final weighted sum, taken modulo 11, comes out to the expected value (typically 0, once the check digit itself is included) — it has no way of confirming that every individual digit is unchanged, only that the *overall* weighted total is unchanged. If two digits are altered in a way that happens to leave the total weighted sum modulo 11 exactly the same as before (for example, one digit's contribution increases by the same amount that another digit's contribution decreases, given their respective weights), the check-digit calculation will still produce the expected result and the error will pass through completely undetected, even though the code itself is now wrong. This is a known limitation of simple check-digit schemes: they catch the overwhelming majority of real-world single-digit typing and scanning mistakes very reliably, but they cannot guarantee detection of every possible combination of errors.

Bài tập luyện tập

A wireless door-lock receiver checks incoming command messages using a checksum, but has *no* ARQ (retransmission) capability built in. Explain the practical limitation this creates when a checksum mismatch occurs.

Lời giải chi tiết

The checksum can only **detect** that an error occurred in the received message; on its own, it defines no automatic mechanism for requesting or triggering a retransmission of the corrupted data. Without ARQ, a corrupted command is most likely simply **discarded or rejected** once the checksum fails to match, and the sending device (or the user) has no automatic way of knowing this happened or of getting the correct data resent — the operation (e.g. unlocking the door) may simply fail silently, requiring the user to notice the failure and manually retry the action themselves.

Bài tập luyện tập

State *one* similarity and *one* difference between an **echo check** and a **checksum** as methods of error detection.

Lời giải chi tiết

Similarity: both methods allow the receiver (or, in the case of an echo check, ultimately the sender) to compare data against what was originally sent, in order to detect whether any part of it was corrupted during transmission.
Difference: a checksum requires only a small calculated value to be sent alongside the original data (little extra traffic), whereas an echo check requires the *entire* block of data to be transmitted a second time in the reverse direction, which roughly doubles the total network traffic needed compared to sending the checksum-based check.

Bài tập luyện tập

A 2-row, 8-column block uses even parity in both directions. The two rows, as originally transmitted, are:

Data bits								Row parity
1	0	1	1	0	1	0	0	0
0	1	0	0	1	0	1	1	0

During transmission, exactly *four* bits are corrupted: both bits in column 2 (rows 1 and 2) and both bits in column 6 (rows 1 and 2) are flipped. (a) Write out the two received rows. (b) Show whether the row and column parity checks detect this error. (c) Explain what this demonstrates about the limits of two-dimensional parity.

Lời giải chi tiết

(a) Row 1 originally `1 0 1 1 0 1 0 0`: flipping column 2 ($0 \to 1$) and column 6 ($1 \to 0$) gives `1 1 1 1 0 0 0 0`. Row 2 originally `0 1 0 0 1 0 1 1`: flipping column 2 ($1 \to 0$) and column 6 ($0 \to 1$) gives `0 0 0 0 1 1 1 1`.

(b) **Row check:** Row 1 received `11110000` has 4 ones (even) — matches stored row parity 0 — **passes, undetected**. Row 2 received `00001111` also has 4 ones (even) — matches stored row parity 0 — **passes, undetected**.
Column check: column 2 received values are row1$= 1$, row2$= 0$, sum $= 1$ (odd), matching the original (odd) column parity for column 2 — **passes**. Column 6 received values are row1$= 0$, row2$= 1$, sum $= 1$ (odd), also matching its original column parity — **passes**. Every other column is unaffected. **Every single row and column check passes**, even though 4 bits are actually wrong.

(c) This demonstrates that two-dimensional parity, like simple one-dimensional parity, can still be fooled by a carefully-positioned *even* number of errors: when exactly the four bits at the corners of a "rectangle" (two specific rows × two specific columns) are all flipped, each affected row loses and gains one 1-bit (no net change to its parity) and each affected column does the same — so no row or column check ever fails, and the error passes through completely undetected. Two-dimensional parity greatly improves on simple parity by *locating* the great majority of realistic single-bit errors, but — like all parity-based schemes — it is not proof against every possible pattern of multiple-bit errors.

Bài tập luyện tập

A network protocol first checks every incoming block of data locally using two-dimensional parity; only if a block cannot be corrected this way (e.g. more than one bit appears to be wrong) does it fall back on a full ARQ retransmission request. Explain why this combination can reduce overall network traffic compared to relying on ARQ alone for every single detected error.

Lời giải chi tiết

Two-dimensional parity can **locate and correct** the great majority of transmission errors — specifically, any single-bit error within a block — using only the parity bits that were already transmitted with the block, *without* needing any further communication with the sender at all. If a system relied on ARQ alone, *every* detected error, including simple single-bit errors, would require a full round trip: a NAK sent back to the sender, followed by the sender retransmitting the *entire* block. Because single-bit errors are corrected locally and immediately by two-dimensional parity, ARQ's more expensive NAK-and-retransmit cycle is only needed for the rarer, more serious cases that two-dimensional parity cannot fix by itself (such as multiple-bit errors within the same block) — reducing the average number of retransmissions, and therefore the average network traffic and delay, needed to deliver each block successfully.

Bài tập luyện tập

Explain why practical ARQ implementations label each transmitted block with a **sequence number**, in addition to using ACK/NAK replies and a timeout timer.

Lời giải chi tiết

A timeout can fire even when the original block *did* arrive correctly, if only the ACK travelling back to the sender was lost or delayed; in that case the sender's automatic retransmission sends the receiver a block it already has. Without any way to tell blocks apart, the receiver could not tell this retransmitted copy apart from a genuinely new block, and might accept the same data twice or place it in the wrong position in the overall message. A **sequence number** attached to each block lets the receiver recognise and **discard duplicate** blocks that arrive a second time, and also lets it notice if an expected sequence number never arrives at all (a block lost completely) — something the ACK/NAK/timeout mechanism on its own cannot guarantee.

Bài tập luyện tập

Why is a Caesar cipher considered insecure regardless of how long the intercepted ciphertext message is?

(A) Because longer messages always contain typing errors

(B) Because there are only 25 possible non-zero shift keys in total, so every key can be tried almost instantly regardless of message length

(C) Because long messages cannot be encrypted using a shift cipher at all

(D) Because the ciphertext automatically reveals the key after 25 characters

Lời giải chi tiết

The total number of possible keys in a Caesar cipher (25, ignoring the trivial shift of 0) does not depend on how long the message is — a computer can simply try decrypting the ciphertext with every one of the 25 shifts and see which one produces readable text, a search that takes a negligible amount of time regardless of whether the message is 5 characters or 5000. **(B)**.

Hardware

Mục tiêu Unit: Hiểu kiến trúc von Neumann và chu trình fetch-decode-execute; phân tích các yếu tố ảnh hưởng đến hiệu năng CPU; phân biệt hệ thống nhúng (embedded system) với máy tính đa năng; nắm vững bộ nhớ chính (RAM/ROM/cache) và bộ nhớ ảo (virtual memory); so sánh các thiết bị lưu trữ thứ cấp; nhận diện và lựa chọn đúng thiết bị nhập/xuất phù hợp cho từng tình huống thực tế.

3.1 The central processing unit (CPU)

Vai trò của CPU

The **central processing unit (CPU)** is the part of a computer that fetches, decodes and executes program instructions stored in main memory. It performs all arithmetic, logical and control operations needed to run software, and coordinates the movement of data between memory, storage and input/output devices.

Nearly all general-purpose computers, from smartphones to supercomputers, are built around the **von Neumann architecture**, a design first described in the 1940s. Its defining feature is that **both the program instructions and the data they operate on are stored together in the same main memory**, and are fetched from that memory using the same set of buses.

Kiến trúc von Neumann

Key characteristics of von Neumann architecture:

- A single main memory holds **both instructions and data** — the CPU cannot tell, just by looking at a memory address, whether the bit pattern stored there is an instruction or a piece of data; this is determined only by *when* it is fetched (during the fetch stage it is treated as an instruction).
- Instructions are executed **one at a time, sequentially**, following the order in which they are stored, unless a jump/branch instruction changes the flow.
- The CPU and memory are connected by three groups of wires called **buses**:
  - **address bus** — unidirectional; carries the memory address the CPU wants to read from or write to.
  - **data bus** — bidirectional; carries the actual instruction or data value between CPU and memory.
  - **control bus** — carries control signals such as read/write, clock timing and interrupt signals.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kiến trúc von Neumann là mô hình máy tính trong đó **chương trình (lệnh) và dữ liệu được lưu chung trong một bộ nhớ chính duy nhất**, thay vì tách riêng thành hai bộ nhớ độc lập. CPU không thể phân biệt một ô nhớ đang chứa lệnh hay dữ liệu chỉ bằng cách nhìn vào giá trị nhị

phân — nó chỉ biết đó là lệnh vì đang lấy nó ra *ở giai đoạn fetch* của chu trình. Ba loại bus kết nối CPU với bộ nhớ: **address bus** (một chiều, mang địa chỉ ô nhớ), **data bus** (hai chiều, mang giá trị thực sự được đọc/ghi), và **control bus** (mang tín hiệu điều khiển như đọc/ghi, xung nhịp, ngắt).

3.1.1 Block diagram of the CPU

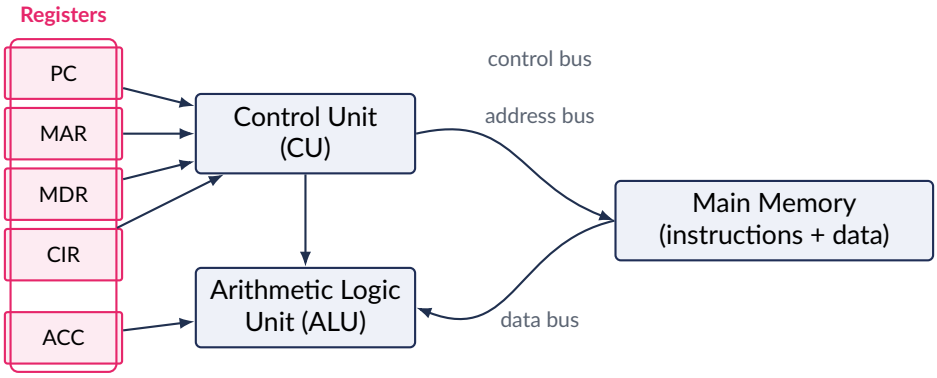


Figure 3.1: Simplified block diagram of von Neumann architecture: the CPU (control unit, ALU, registers) connected to main memory by address, data and control buses.

Key registers and their roles

- **Program Counter (PC)** — holds the memory address of the *next* instruction to be fetched.
- **Memory Address Register (MAR)** — holds the memory address currently being accessed (read from or written to).
- **Memory Data Register (MDR)** — holds the actual instruction or data value that has just been transferred to/from memory.
- **Current Instruction Register (CIR)** — holds the instruction currently being decoded and executed.
- **Accumulator (ACC)** — holds the intermediate results of arithmetic and logical operations performed by the ALU.

VN **GIẢI THÍCH TIẾNG VIỆT**

PC (bộ đếm chương trình) giữ địa chỉ của lệnh *tiếp theo* sẽ được lấy ra — không phải lệnh hiện tại. **MAR** giữ địa chỉ ô nhớ đang được truy cập. **MDR** giữ giá trị (lệnh hoặc dữ liệu) vừa được chuyển giữa CPU và bộ nhớ. **CIR** giữ lệnh hiện đang được giải mã/thực thi. **ACC** giữ kết quả trung gian của các phép tính số học/logic. Học sinh dễ nhầm PC với CIR — PC luôn "đi trước một bước" so với lệnh đang thực thi.

3.2 The fetch--decode--execute cycle

Ba giai đoạn của mỗi lệnh

Every single instruction that the CPU runs goes through the same three stages, known as the **fetch–decode–execute cycle**:

1. **Fetch** — the instruction addressed by the PC is copied from main memory into the CPU.
2. **Decode** — the control unit interprets the instruction to work out which operation it represents and what operands it needs.
3. **Execute** — the operation is actually carried out (e.g. an addition in the ALU, or a value being stored back to memory).

Chi tiết giai đoạn Fetch

The fetch stage can be broken down into these precise steps:

1. The address held in the **PC** is copied into the **MAR** (this is the address of the instruction to be fetched).
2. The **PC** is incremented (increased by 1, or by the instruction length) so that it now points to the *next* instruction — this happens during fetch, *before* the current instruction has even been executed.
3. The address in the MAR is placed on the **address bus**, and a 'read' signal is placed on the control bus.
4. The content of the memory location addressed is placed on the **data bus** and copied into the **MDR**.
5. The content of the MDR is copied into the **Current Instruction Register (CIR)**, ready to be decoded.

(!) Lỗi thường gặp: Many students assume the PC is incremented *after* the instruction has executed. In fact the PC is incremented immediately during the **fetch** stage, straight after its old value has been copied into the MAR. This matters because a jump/branch instruction, if it changes the PC during the **execute** stage, must overwrite this already-incremented value — otherwise the branch would not work correctly.

Chi tiết giai đoạn Decode và Execute

Once the fetch stage has copied an instruction into the CIR, the cycle continues:

- **Decode** — the **control unit (CU)** examines the bit pattern in the CIR and splits it into two parts: an **opcode** (which identifies the operation to be performed, e.g. "load", "add", "store", "jump") and an **operand** (which identifies the data or address the operation should act on). The CU uses the opcode to work out which internal control signals must be generated to carry out that particular operation, and decodes the operand into the actual memory address or register involved.
- **Execute** — the control unit sends out the control signals decoded above, which cause the required action to actually take place. This might mean: the ALU performing an arithmetic operation (e.g. addition) using the accumulator and a value fetched from memory; a value being copied from the accumulator into a specified memory address (a "store" instruction);

or, for a jump/branch instruction, a new value being written directly into the PC so that the next fetch begins from a different point in the program rather than the next sequential address.

Once execute is complete, the whole cycle immediately repeats: the CPU fetches the next instruction using the (already-updated) value in the PC.

VN **GIẢI THÍCH TIẾNG VIỆT**

Giai đoạn **Decode** (giải mã): bộ điều khiển (CU) tách lệnh trong CIR thành **mã lệnh (opcode)** — cho biết phép toán gì (nạp, cộng, lưu, nhảy...) — và **toán hạng (operand)** — cho biết dữ liệu/địa chỉ liên quan. Giai đoạn **Execute** (thực thi): CU phát tín hiệu điều khiển để thực sự thực hiện phép toán đó — có thể là ALU tính toán, giá trị được lưu vào bộ nhớ, hoặc với lệnh nhảy (jump), giá trị mới được ghi thẳng vào PC để chu trình fetch tiếp theo bắt đầu từ một vị trí khác thay vì địa chỉ kế tiếp theo thứ tự thông thường.

3.2.1 Register trace: following values through three instructions

To understand the cycle fully it helps to trace the exact contents of each register, instruction by instruction. Suppose main memory contains the following (simplified, made-up) instructions, where each memory address holds one instruction or one data value:

Address	Content	Meaning
100	LDA 200	Load the value at address 200 into the ACC
101	ADD 201	Add the value at address 201 to the ACC
102	STA 202	Store the value in the ACC into address 202
200	15	data value
201	7	data value
202	(empty)	result will be stored here

Assume the PC starts at 100, and the ACC starts at 0. The table below traces the exact value in each register after every micro-step.

Instr.	Step	PC	MAR	MDR	CIR	ACC
LDA 200	start	100	–	–	–	0
LDA 200	fetch: PC→MAR, PC+1	101	100	–	–	0
LDA 200	fetch: mem[100]→MDR	101	100	LDA200	–	0
LDA 200	fetch: MDR→CIR	101	100	LDA200	LDA200	0
LDA 200	execute: mem[200]→ACC	101	200	15	LDA200	15
ADD 201	fetch: PC→MAR, PC+1	102	101	15	LDA200	15
ADD 201	fetch: mem[101]→MDR	102	101	ADD201	LDA200	15
ADD 201	fetch: MDR→CIR	102	101	ADD201	ADD201	15
ADD 201	execute: mem[201]→MDR	102	201	7	ADD201	15
ADD 201	execute: ACC ← ACC+MDR	102	201	7	ADD201	22
STA 202	fetch: PC→MAR, PC+1	103	102	7	ADD201	22
STA 202	fetch: mem[102]→MDR	103	102	STA202	ADD201	22
STA 202	fetch: MDR→CIR	103	102	STA202	STA202	22
STA 202	execute: ACC→MDR	103	202	22	STA202	22
STA 202	execute: MDR→mem[202]	103	202	22	STA202	22

VN **GIẢI THÍCH TIẾNG VIỆT**
Bảng trên mô phỏng chính xác từng bước một khi CPU chạy 3 lệnh liên tiếp: `LDA 200` (nạp giá trị tại địa chỉ 200 vào ACC), `ADD 201` (cộng thêm giá trị tại địa chỉ 201 vào ACC), `STA 202` (lưu kết quả trong ACC vào địa chỉ 202). Chú ý: **PC luôn tăng lên trước** (ngay trong giai đoạn fetch của mỗi lệnh) — ví dụ khi đang fetch lệnh tại 100, PC đã nhảy lên 101 ngay cả trước khi lệnh 100 được thực thi. Kết quả cuối cùng: $15 + 7 = 22$ được lưu vào địa chỉ 202 — khớp với phép tính số học đơn giản, xác nhận bảng truy vết là chính xác.

3.3 Factors affecting CPU performance

Ba yếu tố chính

The overall performance (speed) of a CPU is influenced mainly by three factors: **clock speed**, **number of cores**, and **cache size**.

- **Clock speed** — measured in gigahertz (GHz), this is the number of fetch–execute cycles (clock pulses) the CPU can carry out per second. A CPU running at 3.6 GHz completes 3.6 billion clock cycles per second. A higher clock speed generally means instructions are processed faster, *provided the architecture is otherwise identical*.
- **Number of cores** — a core is an independent processing unit within the CPU capable of fetching and executing its own stream of instructions. A **multi-core** processor (e.g. dual-core, quad-core, octa-core) can, in principle, execute several instruction streams simultaneously (in parallel), rather than one at a time.
- **Cache size** — a larger cache means more frequently/recently used instructions and data can be stored very close to the CPU, reducing the number of slow accesses to main memory (RAM).

VN **GIẢI THÍCH TIẾNG VIỆT**
Ba yếu tố ảnh hưởng đến hiệu năng CPU: **tốc độ xung nhịp** (clock speed, đơn vị GHz — số chu trình lệnh thực hiện được mỗi giây), **số nhân** (cores — mỗi nhân có thể xử lý một luồng lệnh độc lập, cho phép xử lý song song), và **kích thước bộ nhớ đệm** (cache — cache lớn hơn giữ được nhiều dữ liệu/lệnh thường dùng gần CPU hơn, giảm số lần phải truy cập RAM chậm hơn).

Factor	Effect of increasing it	Limitation
Clock speed (GHz)	More clock cycles per second, so more fetch–execute cycles can be completed per second on a single core	Generates more heat; cannot be increased indefinitely without overheating or excessive power use
Number of cores	Multiple independent instruction streams can be processed at the same time (in parallel)	Only helps if the software/task can actually be divided into independent parts; unhelpful for strictly sequential tasks
Cache size	More frequently/recently used data and instructions stay close to the CPU, reducing slow trips to RAM	Larger cache is more expensive and can take marginally longer to search; benefit depends on how much data is reused

(!) Lỗi thường gặp: More cores do **not** automatically make every task faster. Parallel processing across multiple cores only speeds up a task if the software has been written to **split its**

work into independent parts that can run simultaneously (e.g. video rendering, scientific simulations). A task that must be performed as one strict sequence of steps (e.g. many simple everyday applications, or a single-threaded task) will gain little or nothing from extra cores — in that case, a higher clock speed on fewer cores may actually perform better.

So sánh hai cấu hình CPU

Consider two CPUs being compared for use in a school computer that mostly runs word processing, web browsing, and occasionally exports one long video for the school's website:

	CPU A	CPU B
Clock speed	4.0 GHz	3.0 GHz
Cores	2	8
Cache size	4 MB	16 MB

Analysis: For everyday tasks such as word processing and web browsing, which are largely single-threaded (one main sequence of instructions), **CPU A's higher clock speed** (4.0 GHz vs 3.0 GHz) will make those everyday tasks feel noticeably snappier, since more cycles per second are being completed on the one thread that matters. However, for the occasional **video export**, a task that video-editing software typically splits across many parallel threads (e.g. different frames or filters processed at once), **CPU B's 8 cores** allow that work to be divided up and processed simultaneously, likely finishing the export faster overall despite the lower clock speed per core. CPU B's larger 16 MB cache also helps by keeping more of the video data close to the CPU, reducing slow RAM accesses during that intensive task. *Conclusion:* the "better" CPU depends on the software actually being run — CPU A suits mostly sequential everyday use, CPU B suits parallelisable heavy workloads.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ví dụ trên so sánh hai cấu hình CPU: CPU A (xung nhịp cao hơn, ít nhân hơn, cache nhỏ hơn) và CPU B (xung nhịp thấp hơn, nhiều nhân hơn, cache lớn hơn). Với các tác vụ thông thường chạy tuần tự (soạn thảo văn bản, duyệt web), CPU A nhanh hơn nhờ xung nhịp cao. Nhưng với tác vụ có thể chia nhỏ và xử lý song song (xuất video), CPU B tận dụng được 8 nhân để xử lý đồng thời nhiều phần việc, nên nhanh hơn dù xung nhịp thấp hơn. Bài học quan trọng: **không có CPU nào "tốt hơn" tuyệt đối** — hiệu năng thực tế phụ thuộc vào việc phần mềm có được thiết kế để tận dụng nhiều nhân hay không.

3.4 Embedded systems

Định nghĩa hệ thống nhúng

An **embedded system** is a computer system built into a single device to perform one dedicated function (or a small, fixed set of functions), rather than being a general-purpose computer capable of running many different types of software chosen by the user.

Typical examples of embedded systems include:

- the microcontroller in a **washing machine**, which controls the wash cycle, water level, temperature and spin speed;
- the **engine management system** in a car, which continuously monitors sensors and adjusts fuel injection and ignition timing;

- the processor in a **digital camera**, which handles image capture, focus, and storage to a memory card;
- microcontrollers inside microwave ovens, digital watches, traffic-light controllers, and many other household or industrial appliances.

Embedded system vs. general-purpose computer

- An embedded system's software (often called **firmware**) is typically fixed and stored permanently in **ROM**, and is not intended to be changed by the end user.
- A general-purpose computer (e.g. a desktop PC, laptop, or smartphone) can run a wide variety of software chosen and installed by the user, with its operating system and applications usually stored in changeable secondary storage.
- Embedded systems are usually cheaper, smaller, use less power, and are optimised for exactly one job; general-purpose computers are more flexible but more complex, larger, and typically more power-hungry.

Characteristic	Embedded system	General-purpose computer
Purpose	One dedicated function (or a small fixed set)	Runs a wide variety of user-chosen software
Software	Fixed firmware, usually stored in ROM, not user-editable	Operating system and applications on changeable secondary storage, freely installed/updated
Cost & size	Usually cheap, small, low power	Usually more expensive, larger, more power-hungry
Flexibility	Very limited — designed for exactly one job	Highly flexible — new software can be added anytime
Example	Washing-machine controller, car engine management system, digital camera processor	Desktop PC, laptop, smartphone

VN **GIẢI THÍCH TIẾNG VIỆT**

Hệ thống nhúng (embedded system) là một hệ thống máy tính được tích hợp bên trong một thiết bị để thực hiện **một chức năng chuyên biệt** (hoặc một số ít chức năng cố định) — ví dụ bộ điều khiển máy giặt, hệ thống quản lý động cơ ô tô, hay bộ xử lý trong máy ảnh kỹ thuật số. Phần mềm của hệ thống nhúng (gọi là *firmware*) thường được ghi cố định trong **ROM** và người dùng không thể (hoặc hiếm khi) thay đổi. Ngược lại, máy tính đa năng (PC, laptop, điện thoại thông minh) có thể chạy nhiều loại phần mềm khác nhau do người dùng tự cài đặt.

Nhận diện hệ thống nhúng

Scenario: A digital thermostat built into a smart oven controls only the oven's heating element based on a temperature sensor; a separate tablet computer in the kitchen is used by the family to browse recipes, watch videos and check email.
Analysis: The oven's thermostat controller is an **embedded system** — it performs exactly one dedicated task (regulating oven temperature), its control software is fixed in ROM, and the user cannot install new applications on it. The tablet, in contrast, is a **general-purpose computer** — it can run many different applications selected by the user (recipe apps, video players, email clients), and its software can be updated, replaced or added to freely.

3.5 Memory

3.5.1 RAM, ROM and cache

RAM vs ROM vs Cache

- **RAM (Random Access Memory)** is the computer's main working memory. It is **volatile** — its contents are lost when power is removed — and it temporarily holds the operating system, currently running applications, and the data they are working with.
- **ROM (Read-Only Memory)** is **non-volatile** — its contents are retained without power — and typically stores the small program needed to start the computer (e.g. the BIOS/bootstrap loader), which cannot normally be changed by the user.
- **Cache** is a small, very fast area of memory situated between the CPU and main memory (or built directly into the CPU chip). It stores copies of the most frequently or recently used instructions and data, so the CPU can retrieve them far more quickly than by fetching them from RAM every time.

There is a fundamental trade-off in memory design: the faster a type of memory is, the more expensive it is per byte, and so **the less of it a computer typically has**. This is why cache (fastest, most expensive) is kept smallest (kilobytes to a few megabytes), RAM is larger but slower (gigabytes), and secondary storage is largest but slowest of all (often terabytes).

Memory level	Typical speed	Typical size	Cost per byte	Volatile?
Registers	Fastest	A handful of bytes	Highest	Yes
Cache	Very fast	KB – a few MB	Very high	Yes
RAM (main memory)	Fast	GB	Moderate	Yes
Secondary storage	Slow	Hundreds of GB – TB	Low	No

This ordering, from the CPU's own registers down to secondary storage, is often called the **memory hierarchy**: as you move down the hierarchy, capacity increases and cost per byte decreases, but speed decreases. Only secondary storage is non-volatile; every level above it loses its contents when the power is switched off.

VN **GIẢI THÍCH TIẾNG VIỆT**

RAM là bộ nhớ chính, **dễ mất dữ liệu** (volatile) khi mất điện, chứa hệ điều hành và các chương trình/dữ liệu đang chạy. **ROM** là bộ nhớ **không mất dữ liệu** (non-volatile) khi mất điện, thường chứa chương trình khởi động máy và người dùng không sửa được. **Cache** là vùng nhớ rất nhỏ và rất nhanh nằm giữa CPU và RAM, lưu tạm các lệnh/dữ liệu được dùng thường xuyên nhất để CPU không phải truy cập RAM (chậm hơn) mỗi lần. Nguyên tắc đánh đổi: bộ nhớ càng nhanh thì càng đắt, nên dung lượng càng nhỏ — đó là lý do cache luôn nhỏ nhất.

3.5.2 Virtual memory

Bộ nhớ ảo là gì

Virtual memory is a memory-management technique used when the RAM installed in a computer is not large enough to hold everything currently needed. The operating system reserves an area of **secondary storage** to act as an extension of RAM.

The process works roughly as follows:

1. Programs and data in RAM are divided into fixed-size blocks called **pages**.

2. When RAM becomes full and a new page needs to be loaded, the operating system identifies pages that have **not been used recently** and copies (**"swaps out"**) them from RAM to the reserved area of secondary storage (sometimes called the **swap file** or **page file**).
3. This frees up space in RAM for the data or instructions that are actively needed right now.
4. If a page that has been swapped out is needed again later, the operating system copies it back (**"swaps in"**) from secondary storage into RAM — if RAM is full at that point, another page must first be swapped out to make room.

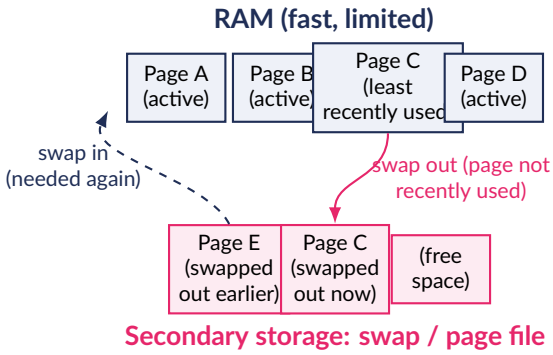


Figure 3.2: Virtual memory (paging): a least-recently-used page (Page C) is swapped out from RAM to secondary storage to free space, while a previously swapped-out page (Page E) can later be swapped back in.

Secondary storage (even fast SSDs) is **much slower** to access than RAM. Virtual memory therefore allows more programs/data to be handled than physical RAM alone would permit, but at the cost of speed whenever swapping occurs. If RAM is so overloaded that the operating system spends most of its time constantly swapping pages in and out rather than doing useful work, this is called **thrashing**, and it causes a severe drop in overall system performance.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bộ nhớ ảo (virtual memory) được dùng khi RAM không đủ chỗ chứa mọi thứ đang cần. Hệ điều hành chia dữ liệu trong RAM thành các khối cố định gọi là **trang (page)**. Khi RAM đầy, hệ điều hành tìm các trang **ít được dùng gần đây nhất** và chuyển ("swap out") chúng sang một vùng dành riêng trên bộ nhớ thứ cấp, giải phóng chỗ cho dữ liệu đang cần dùng ngay. Khi trang đó cần lại, nó được nạp trở lại ("swap in") vào RAM. Vì bộ nhớ thứ cấp chậm hơn RAM rất nhiều, việc swap liên tục làm chậm máy; nếu xảy ra quá thường xuyên gọi là hiện tượng **thrashing** (máy "ì" nặng vì cứ mải swap qua lại thay vì xử lý công việc thật).

Ảnh hưởng của bộ nhớ ảo lên hiệu năng

Scenario: A laptop has 8 GB of RAM. A user opens a web browser with 40 tabs, a video-editing application, and a virtual machine simultaneously, together requiring roughly 14 GB of memory.
Analysis: Since the combined requirement (14 GB) exceeds the physical RAM (8 GB), the operating system relies heavily on virtual memory, continuously swapping pages belonging to background tabs and idle parts of the video editor out to the hard disk/SSD, while keeping the currently active window's pages in RAM. Every time the user switches to a different tab or window, the operating system may need to swap that program's pages back into RAM, evicting something else — causing the noticeable lag many users experience when RAM is close to full.

Adding more physical RAM would reduce how often swapping is needed, directly improving responsiveness.

3.6 Secondary storage

Ba loại lưu trữ thứ cấp

Secondary storage provides non-volatile, long-term storage of data and programs when the computer is switched off. The three main technologies are:

- **Magnetic storage** (e.g. hard disk drives, HDD) — data is stored as magnetised regions on spinning metal platters, read/written by a moving mechanical head.
- **Optical storage** (e.g. CD, DVD, Blu-ray) — data is stored as microscopic pits and lands on a disc, read by a laser.
- **Solid-state storage** (e.g. SSD, USB flash drives, memory cards) — data is stored electronically in flash memory chips, with no moving parts.

Technology	Advantages	Disadvantages	Typical use
Magnetic (HDD)	Large capacity per $ (cost-effective); mature, well-understood technology	Slower access than SSD; moving parts wear out and are fragile if dropped/knocked; noisier, uses more power	Bulk storage on desktop PCs and servers where cost per GB matters more than speed
Optical (CD/DVD/Blu-ray)	Cheap to produce/distribute in bulk; discs are portable and read-only versions resist accidental overwrite	Relatively low capacity; slow access; easily scratched/damaged; drives are becoming less common on modern devices	Distributing software/-media, archival backups, video/audio disc releases
Solid-state (SSD/flash)	Very fast access (no moving parts); durable/shock-resistant; silent; low power use	More expensive per GB than magnetic storage; limited number of write cycles over its lifetime	Laptops, smartphones, portable drives, and any use needing fast access or durability

VN **GIẢI THÍCH TIẾNG VIỆT**

Ba công nghệ lưu trữ thứ cấp: **từ tính** (HDD — đĩa quay, đầu đọc/ghi cơ học, rẻ, dung lượng lớn nhưng chậm và dễ hỏng khi va đập); **quang học** (CD/DVD/Blu-ray — đọc bằng tia laser, rẻ để phân phối hàng loạt nhưng dung lượng thấp, dễ trầy xước); **thể rắn** (SSD/flash — không có bộ phận chuyển động, nhanh, bền, nhưng đắt hơn trên mỗi GB và có giới hạn số lần ghi).

A decision framework for choosing secondary storage. When recommending a storage device for a given scenario, weigh up: (1) **capacity** needed; (2) **access speed** required (how often, and how quickly, must data be read/written); (3) **portability** (does it need to be carried around, and is it exposed to knocks/drops?); (4) **durability** (how long must the data remain safely readable, and under what physical conditions?); and (5) **cost** relative to the budget available. No single storage technology wins on every criterion at once, so the "best" answer always depends on which of these criteria matters most in the specific scenario described.

Chọn thiết bị lưu trữ: kho lưu trữ hồ sơ lịch sử

Scenario: A national library wants to store a digitised archive of historical newspaper records, totalling 40 TB. The records will be written once and read only very rarely (perhaps a few times per year, by researchers), and must remain safely readable for decades. Budget is limited relative to the huge capacity required.
Analysis of options:

- An **SSD** array would offer fast access, but at 40 TB scale would be far more expensive than magnetic storage, and its speed advantage is wasted since the archive is read only occasionally — SSDs are also not primarily chosen for long, unpowered archival life.
- **Optical discs** (e.g. Blu-ray) could work for small archives, but at 40 TB the library would need thousands of individual discs, making management, cataloguing and physical storage impractical.
- **Magnetic tape or HDD-based archival storage** is the recommended choice: it offers the **lowest cost per terabyte** at this scale, and since the data is rarely accessed, the slower access speed of magnetic storage is not a real drawback. Large-capacity magnetic drives can hold the full archive across a manageable number of units.

Conclusion: magnetic storage is recommended, because the low access frequency makes its main weakness (speed) irrelevant, while its main strength (low cost per GB at huge capacities) directly matches the library's requirement.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ví dụ này minh họa nguyên tắc chọn thiết bị lưu trữ: khi dữ liệu **rất ít khi được truy cập** nhưng **dung lượng cực lớn** (40 TB hồ sơ lịch sử), tốc độ không còn là ưu tiên — chi phí trên mỗi GB mới là yếu tố quyết định. Vì vậy lưu trữ từ tính (ổ cứng/băng từ dung lượng lớn) là lựa chọn hợp lý nhất, trong khi SSD (đắt, nhanh nhưng lãng phí vì ít dùng) và đĩa quang (không đủ dung lượng, khó quản lý hàng nghìn đĩa) đều không phù hợp.

Chọn thiết bị lưu trữ: ổ backup di động

Scenario: A freelance photographer needs a single portable device to carry in a camera bag, used daily to back up photos in the field, sometimes dropped or jostled around other equipment, with a moderate capacity requirement (around 2 TB).
Analysis: A traditional portable HDD would offer the needed capacity cheaply, but its spinning platters and moving read/write head make it **vulnerable to damage from drops and vibration** during travel — a real risk for daily field use. A portable **SSD**, although costing more per GB, has **no moving parts**, making it far more shock-resistant and reliable in this rough, mobile environment; it is also faster, so daily backups complete quickly between shoots, and it is smaller and lighter to carry.
Conclusion: the portable SSD is the better recommendation here — the durability and speed

benefits outweigh the higher cost per GB, because the device's daily rough handling in the field is the dominant real-world constraint, not raw storage cost.

3.7 Input devices

Thiết bị nhập

An **input device** allows data to be entered into a computer system from the outside world, converting a physical action, image, sound or measurement into digital data the computer can process.

Device	What it senses / reads	Typical use-case
Keyboard	Individual key presses (letters, numbers, symbols)	Typing documents, entering commands, data entry
Mouse	Movement and button clicks	Pointing, selecting, dragging on a graphical interface
Touchscreen	Direct touch/pressure at screen coordinates	Smartphones, tablets, self-service kiosks, ATMs
Scanner (2D)	Reflected light off a flat image/-document	Digitising paper documents and photographs
Scanner (3D)	Surface shape/geometry of a physical object (laser or structured light)	Reverse engineering, 3D printing preparation, medical imaging
Digital camera	Light intensity/colour across a sensor array	Photography, video conferencing, security surveillance
Microphone	Sound wave vibrations (air pressure changes)	Voice calls, voice assistants, audio/video recording
Sensor – temperature	Ambient or object temperature	Weather stations, greenhouse-/oven control, medical monitoring
Sensor – light	Light intensity	Automatic street lighting, camera auto-exposure, smartphone screen brightness
Sensor – pressure	Physical force/pressure applied	Intruder alarms (weight on a mat), industrial process control
Sensor – proximity	Presence/distance of a nearby object	Automatic doors, parking-assist systems, phone screen-off during calls
Sensor – infrared	Infrared radiation/heat signature	Motion detectors, night-vision, remote controls
Sensor – moisture	Water/humidity content	Automated plant irrigation, weather monitoring
Barcode reader	Reflected laser light off black-/white barcode stripes	Supermarket checkouts, warehouse stock control
RFID reader	Radio-frequency signal from an RFID tag/chip	Contactless access cards, livestock/asset tracking, library book tags
Magnetic stripe reader	Data encoded on a magnetic stripe	Older bank cards, hotel key cards, ID cards
Chip-and-PIN reader	Data on an embedded microchip plus a typed PIN	Secure debit/credit card payments
OMR (optical mark recognition)	Presence/position of pencil/pen marks in predefined boxes	Marking multiple-choice exam answer sheets, lottery tickets, surveys
OCR (optical character recognition)	Shapes of printed/handwritten characters, converted to editable text	Digitising and making printed documents searchable/editable, passport scanning

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng trên liệt kê các thiết bị nhập phổ biến và những gì chúng cảm nhận/đọc được. Cần phân biệt rõ: **OMR** (nhận dạng dấu quang học) chỉ phát hiện *vị trí có dấu tô* (ví dụ ô tròn được tô đen trên phiếu trắc nghiệm) — không đọc được chữ viết. **OCR** (nhận dạng ký tự quang học) thì phức tạp hơn nhiều: nó phân tích *hình dạng của từng ký tự* để chuyển thành văn bản có thể chỉnh sửa được. Các loại cảm biến (sensor) khác nhau — nhiệt độ, ánh sáng, áp suất, khoảng cách (proximity), hồng ngoại, độ ẩm — đều chuyển một đại lượng vật lý thành tín hiệu số.

Cảm biến, ADC và hệ thống giám sát

Sensors measure a physical, real-world quantity (such as temperature, light, or pressure) which naturally varies **continuously** — this is called an **analogue** signal. A computer's CPU, however, can only process **digital** data (discrete binary values). An **analogue-to-digital converter (ADC)** is therefore placed between the sensor and the computer, sampling the analogue signal at regular intervals and converting each sample into a binary number the CPU can process. This combination of sensor + ADC underlies most automated monitoring and control systems, such as the greenhouse or engine-management examples described elsewhere in this chapter: the sensor continuously measures a condition, the ADC digitises the readings, and the CPU compares each digitised reading against a stored threshold value to decide whether an output device (e.g. an actuator) needs to be triggered.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cảm biến đo một đại lượng vật lý biến đổi **liên tục** (analogue) — ví dụ nhiệt độ tăng giảm mượt mà theo thời gian, không phải theo bước nhảy rời rạc. Nhưng CPU chỉ xử lý được dữ liệu **số (digital)** — các giá trị nhị phân rời rạc. Vì vậy cần một bộ chuyển đổi **ADC (analogue-to-digital converter)** đặt giữa cảm biến và máy tính: ADC lấy mẫu tín hiệu analogue theo chu kỳ đều đặn và chuyển mỗi mẫu thành một số nhị phân để CPU xử lý được. Đây chính là cơ chế nền tảng của các hệ thống giám sát/điều khiển tự động (như ví dụ nhà kính tự động ở phần bài tập).

Chọn thiết bị nhập phù hợp: kiểm kho tự động vs chấm thi trắc nghiệm

Scenario A – automated warehouse stock-taking: A logistics company needs to quickly and accurately count thousands of individually labelled boxes as they move through a warehouse, updating stock levels automatically without a worker typing anything.
Scenario B – marking multiple-choice exam sheets: An exam board needs to mark tens of thousands of paper answer sheets, where each question is answered by shading one of several bubbles (A, B, C or D).
Analysis: For Scenario A, a **barcode reader** (or RFID reader, if tags are attached) is the appropriate device: each box carries a unique barcode/tag that can be scanned in an instant as it passes a reader, directly linking the physical item to its stock record with no manual typing and very low error rate. An OMR device would be completely unsuitable here, since boxes do not have shaded bubbles to detect. For Scenario B, an **OMR (optical mark recognition)** device is the appropriate choice: it is specifically designed to detect which bubble on a standard-format answer sheet has been shaded, processing very large volumes of sheets rapidly and consistently. A barcode reader would be unsuitable here, since exam sheets do not carry barcodes representing the student's chosen answers.
Conclusion: the "best" input device always depends on the exact nature of the data being captured — a barcode/RFID reader suits uniquely-tagged physical items moving at speed, while OMR suits standardised mark sheets, and the two are not interchangeable.

3.8 Output devices

Thiết bị xuất

An **output device** converts digital data processed by the computer into a form a human (or another machine) can perceive or use — visual, audible, printed, or physical/mechanical.

Device	How it produces output	Typical use-case
Monitor (LCD/LED)	Illuminates/filters an array of tiny pixels to display an image	General-purpose visual display for any computer
Inkjet printer	Sprays tiny droplets of liquid ink onto paper	Home/small-office printing, occasional colour photo printing
Laser printer	Uses a laser to charge a drum, which attracts toner powder fused onto paper by heat	High-volume office printing, fast black-and-white text documents
3D printer	Deposits/cures material layer by layer to build a physical object	Prototyping, custom manufacturing, medical/dental models
Plotter	Moves a pen (or cutting blade) precisely across large sheets to draw continuous lines	Architectural/engineering drawings, large-format vinyl cutting
Speaker	Vibrates a diaphragm to reproduce sound waves from an electrical signal	Music/video playback, voice assistants, alarms
Actuator	Converts an electrical control signal into physical movement (motor, valve, etc.)	Robotics, automatic doors, industrial machinery, car engine controls

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng trên liệt kê các thiết bị xuất phổ biến. Cần phân biệt **máy in phun (inkjet)** — phun các giọt mực lỏng cực nhỏ, chất lượng màu tốt nhưng chậm hơn và tốn mực với khối lượng lớn — với **máy in laser** — dùng tia laser tích điện lên trống rồi hút bột mực (toner) và nung chảy lên giấy, in nhanh và bền với khối lượng lớn, đặc biệt phù hợp in văn bản đen trắng số lượng lớn. **Actuator** là thiết bị chuyển tín hiệu điện thành *chuyển động vật lý thực sự* (ví dụ động cơ, van) — khác với các thiết bị xuất chỉ hiển thị hoặc phát âm thanh.

A **3D printer** builds a solid physical object through **additive manufacturing**: it reads a digital 3D model and deposits or cures material (such as molten plastic filament, resin, or powder) one thin horizontal layer at a time, with each new layer fusing to the one below, until the complete three-dimensional object has been built up from the ground upward. This differs fundamentally from a standard 2D printer, which only ever places ink or toner onto a single flat sheet of paper.

DAC: chiều ngược lại của ADC

Just as sensors need an ADC to convert their continuous analogue readings into digital data the CPU can process, the reverse conversion is needed to control many physical output devices. A **digital-to-analogue converter (DAC)** converts a digital (binary) control signal from the CPU into a continuously varying analogue signal, which can then be used to drive an **actuator** — for example, smoothly varying the voltage sent to a motor so that a car engine's fuel valve opens to a precise, continuously variable position, rather than being limited to a small number of fixed on/off states.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ngược với ADC, **DAC (digital-to-analogue converter)** chuyển tín hiệu số từ CPU thành tín hiệu analogue liên tục để điều khiển các thiết bị vật lý như động cơ hay van — ví dụ điều chỉnh độ mở van nhiên liệu động cơ ô tô một cách mượt mà, liên tục, thay vì chỉ có vài trạng thái bật/tắt cố định.

Chọn thiết bị xuất phù hợp: máy in trường học vs máy in cá nhân

Scenario A – school office: A busy school office needs to print several hundred pages of black-and-white text documents (letters, timetables, exam papers) every single day, as quickly and cheaply per page as possible.
Scenario B – hobbyist photographer: A hobbyist occasionally wants to print a handful of high-quality colour photographs at home, perhaps only once every few weeks.
Analysis: For Scenario A, a **laser printer** is clearly the better choice: it prints large volumes of text far faster than an inkjet printer, and its running cost per page for black-and-white text is significantly lower, since toner cartridges print many more pages before needing replacement compared with the ink cartridges an inkjet would consume at this volume. For Scenario B, an **inkjet printer** is more appropriate: inkjets generally produce smoother colour gradients and finer detail on photographic paper, and since the hobbyist prints only occasionally, the higher up-front cost of a laser printer (and the risk of toner drying out from infrequent use, versus ink cartridges which are cheaper to replace occasionally) is not justified.
Conclusion: laser printers suit high-volume, mostly monochrome/text printing where speed and low cost-per-page matter most; inkjet printers suit lower-volume printing where colour photo quality matters more than speed or running cost.

3.9 Practice questions

Bài tập luyện tập

Put the following steps of the fetch stage into the correct order:
(i) The content of the addressed memory location is placed on the data bus and copied into the MDR.
(ii) The content of the MAR is placed on the address bus.
(iii) The content of the PC is copied into the MAR.
(iv) The PC is incremented.
(v) The content of the MDR is copied into the CIR.

Lời giải chi tiết

The correct order is: (iii) → (iv) → (ii) → (i) → (v).
First the PC's address is copied into the MAR (iii); the PC is then immediately incremented so it is ready to point to the next instruction (iv); the MAR's address is placed on the address bus so the memory location can be located (ii); the content at that address is fetched via the data bus into the MDR (i); finally the MDR's content is copied into the CIR ready for decoding (v).

Bài tập luyện tập

Which of the following statements correctly distinguishes RAM from ROM?

(A) RAM is non-volatile and ROM is volatile

(B) RAM is volatile and used for temporary working storage; ROM is non-volatile and typically stores start-up instructions

(C) RAM and ROM are both volatile, but RAM is faster

(D) RAM stores the boot program permanently; ROM holds the currently running applications

Lời giải chi tiết

The correct answer is **B**. RAM is volatile (loses its contents when power is removed) and temporarily holds the operating system, running applications and their data. ROM is non-volatile (retains its contents without power) and typically stores the fixed start-up/bootstrap program. Option A reverses the volatility; option C is wrong because ROM is not volatile; option D swaps the roles of RAM and ROM.

Bài tập luyện tập

A video editor working with large 4K video files needs a new laptop and asks whether to choose an HDD or an SSD for the internal storage. Recommend a device and justify your answer.

Lời giải chi tiết

An **SSD** should be recommended. Video editing involves reading and writing very large files repeatedly and often requires scrubbing through footage in real time; an SSD's much faster read/write speed (no moving parts) allows footage to load and preview far more smoothly than an HDD. SSDs are also more durable and shock-resistant, which suits a laptop that may be moved around. Although an SSD costs more per gigabyte than an HDD, the significant gain in workflow speed for video editing outweighs the extra cost.

Bài tập luyện tập

Match each register to its correct function: PC, MAR, MDR, CIR.
(a) Holds the address of the next instruction to be fetched.
(b) Holds the address currently being accessed in memory.
(c) Holds the instruction currently being decoded/executed.
(d) Holds the value just transferred to or from memory.

Lời giải chi tiết

(a) PC (Program Counter); (b) MAR (Memory Address Register); (c) CIR (Current Instruction Register); (d) MDR (Memory Data Register).

Bài tập luyện tập

Explain why cache memory is kept deliberately small rather than being made as large as RAM.

Lời giải chi tiết

Cache uses very fast (and therefore expensive) memory technology, so increasing its size significantly raises manufacturing cost. Also, a very large cache would take longer to search through to check whether the needed data is present, partly offsetting the speed advantage cache is meant to provide. Since cache only needs to hold the small subset of data/instructions used most frequently or recently, keeping it small while RAM (larger but slower and cheaper per byte) handles everything else strikes the best balance between speed and cost.

Bài tập luyện tập

Trace the ACC register for the following sequence of instructions, given ACC starts at 0: LDA 50 (loads value at address 50, which is 12), ADD 51 (adds value at address 51, which is 9), SUB 52 (subtracts value at address 52, which is 5). Give the value of ACC after each instruction.

Lời giải chi tiết

After LDA 50: ACC $= 12$ (the value at address 50 is loaded directly into ACC).
After ADD 51: ACC $= 12 + 9 = 21$ (the value at address 51 is added to the current ACC).
After SUB 52: ACC $= 21 - 5 = 16$ (the value at address 52 is subtracted from the current ACC).
Final value of ACC $= 16$.

Bài tập luyện tập

State the purpose of the address bus, the data bus and the control bus, and explain why the address bus only needs to carry signals in one direction while the data bus needs to carry signals in both directions.

Lời giải chi tiết

The **address bus** carries the memory address that the CPU wants to read from or write to. The **data bus** carries the actual instruction or data value being transferred between the CPU and memory. The **control bus** carries signals that coordinate the transfer, such as read/write signals and timing pulses.
The address bus is unidirectional because only the CPU ever needs to specify *which* location it wants to access — memory itself never needs to send an address back to the CPU. The data bus, however, must be bidirectional because data needs to travel *from* memory to the CPU when reading (e.g. fetching an instruction) and *from* the CPU to memory when writing (e.g. storing a result), so signals must be able to flow both ways along it.

Bài tập luyện tập

Which of the following best describes the von Neumann architecture?

(A) Instructions and data are stored in two completely separate memories, each with its own dedicated bus

(B) Instructions and data are stored together in the same main memory and are fetched using the same buses

(C) Only data is stored in memory; instructions are hard-wired directly into the CPU circuitry

(D) Each core has its own private main memory that no other core can access

Lời giải chi tiết

The correct answer is **B**. In von Neumann architecture, a single main memory stores both program instructions and the data those instructions operate on, and both are transferred to/from the CPU using the same address, data and control buses. Option A describes a Harvard architecture (separate instruction and data memories), option C is incorrect since instructions must be fetched from memory not wired into the CPU, and option D is not a defining feature of von Neumann architecture.

Bài tập luyện tập

A gaming laptop is advertised with a 5.2 GHz clock speed and 6 cores. A rival laptop has a 3.8 GHz clock speed and 16 cores. Explain, with reference to the type of software typically used, why neither laptop can be said to be ”faster” in every situation.

Lời giải chi tiết

Clock speed determines how many fetch–execute cycles a single core can perform per second, so the 5.2 GHz laptop will process a single sequential (single-threaded) task faster per core — this benefits software that cannot be split across multiple cores, such as many older or simpler applications, or a task that depends on the result of the previous step before it can continue. The 16-core laptop, despite its lower clock speed, can divide a suitably parallelisable task (such as rendering a complex 3D scene, encoding video, or running many independent simulations at once) across far more cores simultaneously, likely completing that type of task faster overall. Since real-world performance depends on *whether the software in use is written to exploit multiple cores*, neither specification alone guarantees better performance for every task — the best choice depends on what the user will actually run.

Bài tập luyện tập

Which of these tasks is most likely to benefit significantly from a CPU with more cores, rather than simply a higher clock speed?

(A) Opening a single plain-text document in a basic text editor

(B) Rendering a 3D animated film, where different frames can be calculated independently

(C) Running a simple calculator application performing one calculation at a time

(D) Displaying a static webpage with no video or animation

Lời giải chi tiết

The correct answer is **B**. Rendering a 3D animated film can be split into many independent subtasks (e.g. different frames, or different regions of a single frame) that can each be calculated on a separate core at the same time, so more cores directly reduce the total rendering time. The other options (A, C, D) describe simple, largely sequential tasks that do not naturally divide into independent parallel parts, so they gain little benefit from extra cores and instead respond mainly to higher clock speed.

Bài tập luyện tập

Define what is meant by an ”embedded system”, and explain one way in which an embedded system differs from a general-purpose computer.

Lời giải chi tiết

An embedded system is a computer system built into a device to carry out one dedicated task (or a small fixed set of tasks), rather than being designed to run a wide variety of user-chosen software. One key difference from a general-purpose computer is that an embedded system’s control software (firmware) is typically stored permanently in ROM and cannot be changed by the end user, whereas a general-purpose computer’s software is usually stored on changeable secondary storage and can be freely installed, updated or removed by the user.

Bài tập luyện tập

For each device below, state whether it is most likely to be an embedded system or a general-purpose computer, and justify your answer:
(a) The microcontroller inside a dishwasher that manages the wash cycle.
(b) A tablet computer used by a family to browse the internet, play games and send emails.

Lời giải chi tiết

(a) This is an **embedded system**. It performs one dedicated function — controlling the dishwasher's wash cycle (water flow, temperature, timing) — and its control software is fixed and not intended to be changed or added to by the user.
(b) This is a **general-purpose computer**. It runs many different types of software chosen and installed by the user (web browser, games, email client), and its operating system and applications can be updated, replaced or added to at any time.

Bài tập luyện tập

Explain the purpose of virtual memory, and describe what happens when the operating system needs to load a new page into a full RAM.

Lời giải chi tiết

Virtual memory allows a computer to run more programs, or handle more data, than its physical RAM could hold on its own, by using a reserved area of secondary storage as an extension of RAM. When RAM is full and a new page needs to be loaded, the operating system identifies a page in RAM that has not been used recently, copies (swaps out) it to the reserved area on secondary storage to free up space, and then loads the required page into the space that has just been freed. If that swapped-out page is needed again later, it is copied back (swapped in) from secondary storage into RAM, which may itself require another page to be swapped out first if RAM is still full.

Bài tập luyện tập

What term describes the situation where a computer's performance drops severely because it spends most of its time swapping pages between RAM and secondary storage rather than executing useful instructions?

(A) Caching
(B) Thrashing
(C) Multitasking
(D) Buffering

Lời giải chi tiết

The correct answer is **B**, thrashing. This occurs when RAM is so overloaded that the operating system must constantly swap pages in and out of secondary storage to keep up with demand, leaving little time for actual processing — causing a severe, noticeable slowdown. Caching (A) refers to storing frequently used data close to the CPU for speed, not a performance problem; multitasking (C) is simply running several programs at once; buffering (D) is temporarily storing data (e.g. during streaming) to smooth out transfer speed differences.

Bài tập luyện tập

A computer with 4 GB of RAM is running several large applications that together require 10 GB of memory. Explain why this computer is likely to run noticeably slower than one with 16 GB of RAM running the exact same applications, using the concept of virtual memory in your answer.

Lời giải chi tiết

With only 4 GB of RAM but 10 GB of memory required, the operating system must rely heavily on virtual memory: it will need to swap pages of data out to secondary storage far more often to make room for whatever program the user is currently working with, and swap other pages back in whenever the user switches tasks. Because secondary storage is much slower to access than RAM, this constant swapping introduces significant delay every time a page is needed but is not currently in RAM. The computer with 16 GB of RAM can hold the entire 10 GB requirement in RAM at once, with no swapping needed at all, so it can access all the required data at RAM speed continuously, resulting in noticeably smoother and faster performance.

Bài tập luyện tập

A company needs to store daily backups of its financial database (500 GB per day) so that any day's backup can be restored within minutes if needed, and cost is a secondary concern compared with restore speed. Recommend an appropriate secondary storage technology and justify your choice.

Lời giải chi tiết

An **SSD-based** storage system should be recommended. Since the priority is being able to restore a backup within minutes, the fast read/write speed of solid-state storage (with no mechanical parts to slow down access) is essential — a magnetic HDD or optical storage would take considerably longer to read back 500 GB of data. Although SSDs cost more per gigabyte than magnetic storage, the scenario explicitly states that cost is secondary to restore speed, so the higher cost is justified by the dramatically faster access time when a restore is urgently needed.

Bài tập luyện tập

Explain one advantage and one disadvantage of optical storage (such as DVD) compared with solid-state storage (such as a USB flash drive).

Lời giải chi tiết

Advantage: Optical discs are very cheap to mass-produce, making them cost-effective for distributing identical copies of software or media to large numbers of people (e.g. selling the same film on DVD to thousands of customers).
Disadvantage: Optical discs are read using a laser and are easily scratched or damaged, which can make data unreadable; they also have much lower storage capacity and slower access speeds than a solid-state flash drive, and optical drives are becoming less common on modern computers and laptops.

Bài tập luyện tập

A school wants to mark thousands of multiple-choice test papers automatically without any human re-typing the answers. Which input technology is most appropriate, and why would a barcode reader not be suitable for this task?

Lời giải chi tiết

OMR (optical mark recognition) is the most appropriate technology. It is specifically designed to detect which bubble or box has been shaded in on a standardised answer sheet, allowing the student's chosen answer for every question to be read automatically and very quickly across thousands of sheets. A barcode reader would not be suitable because it is designed to read a printed pattern of parallel lines representing a fixed code (such as a product number) — it has no way of detecting which of several bubbles on an answer sheet has been shaded by a student, since that information is not encoded as a barcode.

Bài tập luyện tập

Which input device would be most appropriate for capturing a customer's signature-free contactless payment using a pre-loaded transport card at a train station gate?

(A) Magnetic stripe reader
(B) RFID reader
(C) OMR
(D) Scanner (2D)

Lời giải chi tiết

The correct answer is **B**, an RFID reader. Modern contactless transport cards contain an embedded RFID chip and antenna; tapping the card near the gate's RFID reader allows data to be exchanged wirelessly over a short range almost instantly, without physical contact — ideal for the high-speed, high-volume flow of passengers through station gates. A magnetic stripe reader (A) requires physically swiping the card through a slot, which is slower and more prone to wear; OMR (C) detects shaded marks on paper, irrelevant here; a 2D scanner (D) reads flat printed images, not embedded chip data.

Bài tập luyện tập

Explain the difference between OCR and OMR, giving one appropriate use for each.

Lời giải chi tiết

OCR (optical character recognition) analyses the shapes of individual printed or handwritten characters on a scanned image and converts them into editable, searchable digital text. A typical use is scanning a printed letter or book page so that its text can be edited in a word processor or searched electronically.
OMR (optical mark recognition) simply detects the presence and position of marks (such as a shaded bubble or tick box) within predefined areas on a form — it does not attempt to recognise character shapes. A typical use is automatically marking multiple-choice exam answer sheets or scanning National Lottery tickets.

Bài tập luyện tập

A greenhouse automation system needs to automatically open a roof vent when the internal temperature rises above a set threshold, and automatically water plants when the soil becomes too dry. Identify one input sensor and one output device this system would need, and explain the role of each.

Lời giải chi tiết

Input sensor: A **temperature sensor** would be needed to continuously monitor the internal temperature of the greenhouse and send readings to the microcontroller, which compares them against the threshold value to decide whether the vent should be opened. A **moisture sensor** would similarly be needed to detect when the soil's water content drops below the required level.
Output device: An **actuator** (e.g. a motor connected to the roof vent, and a separate one controlling a water valve/pump) would be needed to convert the microcontroller's electrical control signal into the actual physical movement of opening the vent or releasing water to the plants.

Bài tập luyện tập

An architecture firm needs to produce large-format technical drawings of building blueprints, up to A0 size, with continuous precise lines rather than a grid of dots. Which output device is most suitable, and why would a standard inkjet printer be a poor choice?

Lời giải chi tiết

A **plotter** is the most suitable output device. Plotters are designed to move a pen (or cutting tool) precisely and continuously across very large sheets of paper, producing accurate, continuous vector lines ideal for technical and architectural drawings, and they can handle oversized paper (such as A0) that standard printers cannot accommodate. A standard inkjet printer would be a poor choice because its maximum paper size is usually far smaller than A0, and it builds an image from a grid of tiny ink dots rather than continuous precise lines, which is less suited to fine technical line work at large scale.

Bài tập luyện tập

Which statement about laser printers compared with inkjet printers is correct?

(A) Laser printers are generally faster and cheaper per page for high-volume black-and-white text printing

(B) Inkjet printers use toner powder fused by heat, while laser printers spray liquid ink droplets

(C) Laser printers cannot print in colour under any circumstances

(D) Inkjet printers are always the better choice regardless of print volume or budget

Lời giải chi tiết

The correct answer is **A**. Laser printers use a laser to charge a drum that attracts toner powder, which is then fused onto the paper by heat; this process is generally faster than inkjet printing and has a lower cost per page for high-volume, mostly black-and-white text printing, since toner cartridges print far more pages than an equivalent ink cartridge. Option B reverses the

two technologies (it is the laser printer that uses toner, and the inkjet that sprays liquid ink); option C is false, as colour laser printers do exist; option D is false, since inkjets are actually better suited to lower-volume, high-quality colour photo printing, not every scenario.

Bài tập luyện tập

Explain why cache memory allows the CPU to run faster overall, and describe what happens if the data the CPU needs is not currently held in the cache.

Lời giải chi tiết

Cache memory is much faster to access than main memory (RAM), so if the instructions or data the CPU needs are already present in the cache, the CPU can retrieve them almost immediately rather than waiting for the comparatively slow process of fetching them from RAM. This is effective because programs tend to reuse the same instructions and data repeatedly over short periods (e.g. inside loops), so keeping recently/frequently used items in cache saves many slow RAM accesses. If the required data is **not** currently in the cache (a "cache miss"), the CPU must fetch it from main memory as normal — this takes longer — and the retrieved data is then usually copied into the cache as well, in case it is needed again soon.

Bài tập luyện tập

State two ways in which increasing a CPU's cache size could still fail to improve overall performance in practice.

Lời giải chi tiết

Two possible reasons: (1) A larger cache is more expensive to manufacture and takes up more space on the CPU chip, so it may not be economically practical beyond a certain point for the intended market/price of the device; (2) A larger cache can take slightly longer to search through when checking whether a given piece of data is present, which can partly offset the speed benefit gained from holding more data close to the CPU, especially if the workload does not actually reuse enough data to benefit from the extra space.

Bài tập luyện tập

A smartwatch contains a small processor that continuously monitors the wearer's heart rate using a built-in sensor and displays the current reading on a small screen; its software cannot be changed by the user beyond installing approved, limited watch-face updates through the manufacturer's own system. Is this best classified as an embedded system or a general-purpose computer? Justify your answer with two points.

Lời giải chi tiết

This is best classified as an **embedded system**, for two reasons: (1) it is dedicated to performing a small, fixed set of specific functions — monitoring heart rate via its sensor and displaying the reading — rather than being designed to run a broad, open-ended range of user-installed applications; (2) the user has no ability to install arbitrary third-party software or change its core control functions, being limited only to manufacturer-approved cosmetic updates (watch faces), which is characteristic of firmware fixed by the manufacturer rather than the freely user-modifiable software found on a general-purpose computer.

Bài tập luyện tập

A supermarket wants to speed up its checkout process by allowing customers to pay using their bank card without inserting it into a slot or typing a PIN, for purchases below a certain value. Which input technology enables this, and how does it differ from a traditional magnetic stripe reader in how it reads the card's data?

Lời giải chi tiết

This is enabled by **RFID / contactless chip technology** (an RFID or NFC reader built into the card terminal). The card contains a small embedded chip and antenna that communicate wirelessly with the reader over a very short range when the card is held near it, allowing the transaction data to be exchanged without any physical contact or insertion. A traditional **magnetic stripe reader**, by contrast, requires the card to be physically swiped through a slot so that the reader's head can directly detect the pattern of magnetised particles encoded on the stripe — it needs physical contact and movement, whereas the contactless method does not.

Bài tập luyện tập

Explain why a digital camera's built-in processor is generally considered an embedded system, and state one component of von Neumann architecture (register, bus, or memory type) that would still be present inside it despite this classification.

Lời giải chi tiết

A digital camera's built-in processor is considered an embedded system because it is dedicated to a fixed set of camera-related functions (capturing images, autofocus, adjusting exposure, saving files to a memory card) rather than running arbitrary general-purpose software chosen by the user, and its core control software is fixed by the manufacturer. Despite being embedded, it is still built around the same fundamental von Neumann concepts as a general-purpose computer: for example, it will still contain a **Program Counter (PC)** to track which instruction of its firmware to fetch next, and use an **address bus** / **data bus** to move instructions and image data between its processor and its internal memory.

Bài tập luyện tập

A company is choosing between two CPUs for its servers, which run software that processes thousands of independent customer requests simultaneously (each request handled separately from the others). CPU X: 3.2 GHz, 4 cores, 8 MB cache. CPU Y: 2.6 GHz, 32 cores, 32 MB cache. Recommend a CPU and justify your choice with reference to the nature of the workload.

Lời giải chi tiết

CPU Y should be recommended. The workload described — thousands of independent customer requests processed separately — is highly parallelisable, since each request does not depend on the others and can be handled on its own separate core. CPU Y's far greater number of cores (32 versus 4) allows many more of these independent requests to be processed simultaneously, which will very likely outweigh its lower per-core clock speed (2.6 GHz versus 3.2 GHz) for this specific type of workload. CPU Y's larger cache (32 MB versus 8 MB) also helps keep more request data close to each core, reducing slow main-memory accesses across so many simultaneous operations. CPU X's higher clock speed would only be the deciding ad-

vantage if the workload were mostly single-threaded/sequential, which is not the case here.

Bài tập luyện tập

A CPU's clock speed is quoted as 2 800 MHz. State this value in GHz, and explain what this number actually represents in terms of the fetch–execute cycle.

Lời giải chi tiết

$2\,800$ MHz $= 2.8$ GHz (since 1 GHz $= 1\,000$ MHz). This value represents the number of clock cycles the CPU can carry out per second: at 2.8 GHz, the CPU's internal clock pulses 2.8 billion times every second, and each pulse can drive a step of the fetch–decode–execute cycle. In practice, a single instruction may take more than one clock cycle to complete (since fetch, decode and execute each require one or more cycles), so the clock speed sets an upper limit on how many instructions can be processed per second rather than being an exact instruction count.

Software

Mục tiêu Unit: Phần mềm hệ thống và phần mềm ứng dụng; chức năng hệ điều hành (quản lý bộ nhớ, đa nhiệm, lập lịch CPU, quản lý tệp tin, quản lý thiết bị ngoại vi, giao diện người dùng, bảo mật, xử lý ngắt); ngắt (interrupts) và độ ưu tiên ngắt; phần mềm tiện ích (diệt virus, chống phân mảnh đĩa, nén tệp, sao lưu dữ liệu); ngôn ngữ lập trình bậc cao và bậc thấp; các loại trình dịch (hợp dịch/biên dịch/thông dịch); và môi trường phát triển tích hợp (IDE) cùng các công cụ gỡ lỗi (breakpoint, chạy từng bước, cửa sổ theo dõi biến).

4.1 System software and application software

All software installed on a computer falls into one of two broad categories. **System software** manages the computer's own hardware and resources, and provides the platform on top of which everything else runs; it is largely invisible to the end user and is not, itself, the reason a person sits down at the computer. **Application software** is software that a user runs directly to accomplish a specific task — writing a document, editing a photo, browsing the web, or playing a game.

Khái niệm cốt lõi

System software includes: the **operating system** (manages hardware and resources for every other program); **utility programs** (housekeeping tools such as antivirus software, disk defragmenters, compression tools and backup software); and **translators** (assemblers, compilers and interpreters, which convert programs written by humans into a form the processor can execute). These exist so that application software *never* has to talk to the physical hardware directly — the OS and its utilities handle that complexity once, safely and consistently, for every application.

Application software includes: word processors, spreadsheets, databases, web browsers, photo/video editors, and games. These exist to let a user perform a specific, real-world task; they rely on the operating system underneath them to access memory, storage, the screen, and input devices.

GIẢI THÍCH TIẾNG VIỆT

VN

Phần mềm hệ thống (system software): quản lý phần cứng và tài nguyên máy tính, làm nền tảng cho mọi phần mềm khác chạy được — gồm **hệ điều hành**, **phần mềm tiện ích** (diệt virus, chống phân mảnh đĩa, nén tệp, sao lưu) và **trình dịch** (hợp dịch, biên dịch, thông dịch). **Phần mềm ứng dụng (application software)**: phần mềm người dùng chạy trực tiếp để hoàn thành một công việc cụ thể — soạn thảo văn bản, bảng tính, trình duyệt web, trò chơi. Phần mềm ứng dụng luôn dựa vào hệ điều hành bên dưới để truy cập bộ nhớ, lưu trữ, màn hình và thiết bị vào.

Ví dụ mẫu · Worked example

Classify each of the following as system software or application software, and justify each answer: (a) Microsoft Word, (b) an antivirus scanner, (c) a Python compiler, (d) a photo-editing app.

(a) **Application software** — the user runs it directly to perform the task of writing/formatting a document.

(b) **System software** (specifically a utility program) — it protects and maintains the computer itself, rather than performing a user's task.

(c) **System software** (a translator) — it converts a user's program into a form the processor can execute; it is a tool that supports *other* software rather than being the end task itself.

(d) **Application software** — the user runs it directly to edit photographs, a specific real-world task.

(!) Lỗi thường gặp: A web browser is application software (the user runs it directly to browse), even though it depends heavily on the operating system's networking and display functions. Do not confuse "depends on system software" with "*is* system software".

4.2 The operating system

The **operating system (OS)** is the single most important piece of system software: it manages a computer's hardware and provides a stable, consistent platform on which every other program runs, so that application developers never need to write code that talks directly to physical memory chips, disk controllers or the screen.

4.2.1 Functions of the operating system

Function	What the OS does	Example scenario
Memory management	Allocates a region of RAM to each running program and keeps programs' memory separate; uses **virtual memory** on secondary storage to simulate extra RAM when physical RAM is full.	Ten browser tabs and a word processor are open at once; the OS gives each its own memory space and swaps rarely-used pages to disk.
Multitasking	Gives the impression that several programs run **simultaneously** by switching the CPU between them extremely rapidly.	Music keeps playing in the background while the user types an email.
Scheduling	Decides *which* waiting process is given the CPU next, and for how long, sharing processor time fairly (or by priority) among all loaded programs.	Three open applications each receive a short CPU time-slice in turn, so all three appear responsive at once.
File management	Organises data into files and folders, and allows programs and users to create, name, move, delete and search for files.	Saving a document under a chosen file name into a chosen folder.
Peripheral/device management	Uses **device drivers** to translate general OS/software commands into instructions a specific physical device understands.	A generic "print" command is converted into signals a particular printer model understands.
User interface	Presents a GUI (windows, icons, menus, pointers) or command-line interface for the user to interact with the computer.	Clicking a folder icon to open it, instead of typing a text command.
Security	Manages user accounts, passwords and access permissions to protect data and resources from unauthorised access.	A student account cannot access administrator-only system files.
Interrupt handling	Detects interrupt signals from hardware/software and organises how and in what order they are serviced (see next section).	A keyboard key-press interrupts a lower-priority background task so it can be dealt with promptly.

The main functions of an operating system.

GIẢI THÍCH TIẾNG VIỆT VN

Hệ điều hành đảm nhận: **quản lý bộ nhớ** (cấp phát RAM riêng cho từng chương trình, dùng **bộ nhớ ảo** khi thiếu RAM); **đa nhiệm** (tạo cảm giác nhiều chương trình chạy cùng lúc); **lập lịch (scheduling)** (quyết định tiến trình nào được dùng CPU tiếp theo và trong bao lâu); **quản lý tệp tin** (tổ chức file/folder); **quản lý thiết bị ngoại vi** (qua driver); **giao diện người dùng**; **bảo mật** (tài khoản, mật khẩu, quyền truy cập); và **xử lý ngắt**.

Ví dụ mẫu · Worked example

A computer has three applications open: a web browser, a spreadsheet recalculating a large formula, and a music player. Explain how **scheduling** and **multitasking** work together here.

The OS's **scheduler** repeatedly decides which of the three waiting programs is given the CPU next and for how long (a short **time-slice**), based on a scheduling policy (e.g. giving each program a turn in rotation). Because each time-slice is extremely short compared to human per-

ception, the CPU rapidly switches between the browser, the spreadsheet calculation and the music player — this rapid switching, made possible by the scheduler's decisions, is what the user experiences as **multitasking**: all three appear to run at the same time, even on a single processor core.

Scheduling and **multitasking** are closely linked but not identical: the **scheduler** is the decision-making function (*which* process runs next, for how long); **multitasking** is the *effect* that decision-making produces (several programs appearing to run at once).

Khái niệm cốt lõi

Two simple scheduling policies illustrate how a scheduler can decide which waiting process runs next:

Round-robin scheduling gives every waiting process an equal, fixed-length time-slice in turn, cycling back to the first process once every process has had a turn — fair, but it does not distinguish an urgent task from a routine one.

Priority-based scheduling instead ranks waiting processes by importance and always gives the CPU to the highest-priority waiting process first — an urgent task is serviced sooner, but a continuous stream of high-priority processes can, in principle, delay a low-priority process indefinitely.

VN **GIẢI THÍCH TIẾNG VIỆT**

Lập lịch round-robin: chia đều mỗi tiến trình một khoảng thời gian CPU bằng nhau, lần lượt xoay vòng — công bằng nhưng không phân biệt việc gấp hay không gấp. **Lập lịch theo độ ưu tiên (priority-based)**: luôn ưu tiên CPU cho tiến trình có độ ưu tiên cao nhất đang chờ — việc gấp được xử lý sớm hơn, nhưng tiến trình ưu tiên thấp có thể bị trì hoãn rất lâu nếu liên tục có tiến trình ưu tiên cao xuất hiện.

4.2.2 File management and security in practice

The OS's **file management** function goes beyond simply storing bytes on a disk: it maintains a directory structure (files organised into folders/directories), keeps track of *where* each file's data physically lives on the storage medium, enforces file naming rules, and lets programs create, open, rename, move, copy and delete files through a consistent set of commands — so that an application never needs to know the low-level details of how the underlying storage hardware works.

The OS's **security** function protects the computer and its data from unauthorised access. This typically includes requiring a **username and password** (or other authentication) before a user can log in, assigning each user account a defined **access level** (e.g. standard user vs. administrator) that determines which files, settings and actions they are permitted to use, and maintaining logs of who accessed what and when.

Ví dụ mẫu · Worked example

A school's shared computer allows students to open and edit their own coursework files, but prevents them from installing new software or viewing another student's private files. Identify the two OS functions responsible, and explain the role of each.

File management is responsible for organising each student's files into their own personal folder and providing the commands (open, save, rename) students use to work with their own coursework.

Security is responsible for restricting what each logged-in account is allowed to do: a standard student account is given a lower **access level** than an administrator account, so the OS blocks software installation (an administrator-only action) and blocks access to files inside another user's private folder, even though the underlying file management system is what actually stores and locates all of those files on the disk.

VN **GIẢI THÍCH TIẾNG VIỆT**

Quản lý tệp tin: tổ chức file theo thư mục, theo dõi vị trí lưu trữ thực tế, cho phép tạo/mở/đổi tên/di chuyển/xoá file qua các lệnh thống nhất. **Bảo mật**: yêu cầu đăng nhập (tên đăng nhập/mật khẩu), gán **mức quyền truy cập** khác nhau cho từng tài khoản (người dùng thường và quản trị viên) để quyết định file/cài đặt/hành động nào được phép — hai chức năng này phối hợp để vừa lưu trữ file hiệu quả, vừa bảo vệ dữ liệu khỏi truy cập trái phép.

4.3 Interrupts

An **interrupt** is a signal sent to the processor — by hardware (e.g. a keyboard, a disk controller, a timer chip) or by software (e.g. an invalid instruction, a request for OS services) — indicating that an event needs the CPU's attention.

Khái niệm cốt lõi

On receiving an interrupt, the CPU:

1. finishes the **current** fetch–execute cycle it is part-way through (it never abandons an instruction half-done);
2. **saves the current register state onto a stack** (so the interrupted program can later resume exactly where it left off);
3. jumps to run a special routine called an **Interrupt Service Routine (ISR)** that deals with the event;
4. once the ISR finishes, **restores the saved state** from the stack and resumes the interrupted program from precisely where it paused.

Interrupts are assigned **priorities**. A **higher-priority** interrupt is allowed to interrupt the servicing of a **lower-priority** one already in progress; a lower-priority interrupt arriving while a higher-priority ISR is running must instead **wait** until that ISR completes.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ngắt (interrupt): tín hiệu (từ phần cứng hoặc phần mềm) báo CPU cần xử lý ngay một sự kiện. CPU: (1) hoàn tất chu trình fetch–execute hiện tại, (2) **lưu trạng thái thanh ghi vào ngăn xếp (stack)**, (3) chạy **ISR (chương trình phục vụ ngắt)**, (4) **khôi phục trạng thái** để tiếp tục chương trình bị ngắt từ đúng chỗ dừng. Ngắt có **độ ưu tiên**: ngắt ưu tiên **cao hơn** có thể chen ngang việc xử lý ngắt ưu tiên thấp hơn đang diễn ra; ngược lại thì phải chờ.

Ví dụ mẫu · Worked example

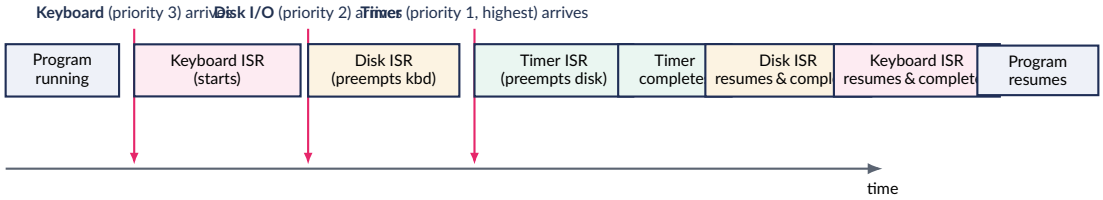
While a document is printing, the user presses a key on the keyboard. Explain how the interrupt is handled.

The keyboard raises a hardware interrupt. The CPU finishes its current fetch–execute cycle, then saves the state of the currently executing (printing) process onto the stack. It jumps to

the keyboard's ISR, which reads and buffers the key press. Once the ISR completes, the CPU restores the saved state and continues the printing task from exactly where it paused.

4.3.1 Competing interrupts of different priorities

In practice, several interrupts can arrive close together while the CPU is still dealing with an earlier one. Priorities decide the order of service, and interrupted ISRs are themselves paused and resumed using the same save-state/restore-state mechanism, nested like a stack.



Nested interrupts (priority 1 = highest): the keyboard interrupt is itself paused by the higher-priority disk interrupt, which is in turn paused by the still-higher-priority timer interrupt. Service completes in **last-interrupted-first-resumed** order: timer, then disk, then keyboard, then the original program.

Ví dụ mẫu · Worked example

While a program is running, three interrupts arrive in quick succession: a **keyboard** interrupt (priority 3, lowest), then — while the keyboard ISR is still running — a **disk I/O** interrupt (priority 2), then — while the disk ISR is still running — a **timer** interrupt (priority 1, highest). Trace the order in which each is serviced, and explain why.

1. The keyboard interrupt (priority 3) arrives first; since nothing higher-priority is currently running, the CPU services it immediately: it saves the program's state and starts the keyboard ISR.
2. While the keyboard ISR runs, the disk I/O interrupt (priority 2) arrives. Because priority 2 is *higher* than the keyboard ISR's priority 3, the disk interrupt is allowed to preempt it: the CPU saves the (partially-completed) keyboard ISR's state onto the stack and starts the disk ISR.
3. While the disk ISR runs, the timer interrupt (priority 1, the highest) arrives. Because priority 1 is higher than the disk ISR's priority 2, it preempts the disk ISR: its state is saved, and the timer ISR runs.
4. Since nothing can outrank priority 1, the **timer ISR runs to completion first**. Its state is discarded (nothing was interrupted *during* it) and the CPU restores the **disk ISR**, which resumes and completes.
5. The CPU then restores the **keyboard ISR**, which resumes and completes.
6. Finally, the CPU restores the **original program**, resuming it from exactly where it was first interrupted.

Service order: timer → disk → keyboard → original program — the reverse of the order in which they were paused, because each was suspended (not abandoned) on top of the one before it, like a stack.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khi nhiều ngắt xảy ra gần nhau, ngắt có **độ ưu tiên cao hơn** sẽ chen ngang việc xử lý ngắt ưu tiên thấp hơn đang chạy dở — trạng thái của ISR đang chạy dở đó cũng được **lưu vào ngăn xếp** y hệt như lưu trạng thái chương trình gốc. Vì thế thứ tự **hoàn tất** luôn ngược với thứ tự **bị tạm dừng**: ngắt ưu tiên cao nhất chạy xong trước, rồi lần lượt khôi phục các ISR bị tạm dừng

theo thứ tự ngược lại, cuối cùng mới quay lại chương trình gốc.

(!) Lỗi thường gặp: Do not assume a higher-priority interrupt cancels or discards the interrupt(s) it preempts. The lower-priority ISR is only **paused** (its state saved), never lost — it always resumes and completes once every higher-priority interruption above it in the stack has finished.

4.4 Utility software

Utility software performs routine "housekeeping" tasks that keep a computer running efficiently, securely and reliably, without directly performing a user's real-world task the way application software does.

Khái niệm cốt lõi

Antivirus software scans files and running programs, comparing them against a database of known **virus signatures** (and using heuristic techniques to spot suspicious behaviour from previously-unseen malware); it **quarantines** or deletes infected files and must be updated regularly so its database includes newly-discovered threats.

Disk defragmentation software reorganises the files stored on a magnetic hard disk drive (HDD) so that each file's data occupies **contiguous** (physically neighbouring) sectors, rather than being scattered in fragments across the disk.

File compression utilities reduce the size of a file (or group of files) for more efficient storage or faster transmission, typically using lossless algorithms so the original file can be exactly reconstructed.

Backup software automatically copies files (or an entire system) to a separate storage location/medium on a regular schedule, so data can be restored if the original is lost, corrupted, or destroyed (e.g. by hardware failure, accidental deletion or a malware attack).

VN GIẢI THÍCH TIẾNG VIỆT

Phần mềm tiện ích: **diệt virus** (quét file/chương trình theo cơ sở dữ liệu chữ ký virus đã biết, cách ly hoặc xoá file nhiễm, cần cập nhật thường xuyên); **chống phân mảnh đĩa** (sắp xếp lại dữ liệu file trên ổ cứng từ tính HDD cho liền mạch); **nén tệp** (giảm kích thước file để lưu trữ/truyền tải, thường không mất dữ liệu); **sao lưu (backup)** (tự động sao chép file/hệ thống sang nơi lưu trữ khác theo lịch để phục hồi khi mất dữ liệu).

4.4.1 Full and incremental backups

Backup software can be configured to perform different types of backup, trading off storage space used against how quickly a backup completes and how quickly data can be restored.

A **full backup** copies *every* selected file, every time it runs. It is simple to restore from (only one backup set is needed), but takes the longest time to perform and uses the most storage space, since unchanged files are copied again and again.

An **incremental backup** copies *only* the files that have changed since the *previous* backup (whether that previous backup was full or incremental). It is much faster to perform and uses far less storage, but restoring the most recent state requires the last full backup *plus every incremental backup* made since, in the correct order.

VN **GIẢI THÍCH TIẾNG VIỆT**

Sao lưu toàn bộ (full backup): sao chép **tất cả** file mỗi lần chạy — phục hồi đơn giản nhưng tốn thời gian và dung lượng. **Sao lưu tăng dần (incremental backup)**: chỉ sao chép các file **đã thay đổi** kể từ lần sao lưu gần nhất — nhanh và tiết kiệm dung lượng hơn, nhưng khi phục hồi cần đến bản sao lưu toàn bộ gần nhất **cộng với** tất cả các bản sao lưu tăng dần sau đó, theo đúng thứ tự.

Ví dụ mẫu · Worked example

A school performs a full backup of its server every Sunday night, then an incremental backup every other night of the week. The server fails on Thursday morning, before that night's backup. Explain which backups are needed to restore the server, and in what order.

The most recent **full backup** (Sunday night) must be restored first, since it is the only backup containing a complete copy of every file. Then each subsequent **incremental backup** — Monday, Tuesday, and Wednesday night — must be restored *in that order*, since each incremental backup only contains the changes made since the backup before it. Restoring them out of order, or skipping one, would leave the server missing some of the week's changes or applying later changes before earlier ones.

4.4.2 Why disk defragmentation does not help an SSD

On a magnetic HDD, as files are created, resized and deleted over time, a single file's data can end up split into many small pieces (**fragments**) stored in non-contiguous, physically scattered sectors. Reading such a file requires the disk's mechanical **read/write head** to physically move back and forth between many different locations on the spinning platter, which takes measurable time (**seek time**). Defragmentation software rewrites the disk so each file's fragments are moved together into contiguous sectors, meaning the read/write head can retrieve the whole file with much less physical movement.

Ví dụ mẫu · Worked example

Explain what disk defragmentation does, and explain why running a defragmentation utility on a solid-state drive (SSD) gives little or no worthwhile benefit.

What it does: on an HDD, defragmentation software identifies files whose data is split into fragments scattered across different physical locations on the platter, and rewrites the disk so that each file's data is stored in **contiguous** sectors. This reduces how far the mechanical read/write head must travel to read a file, which reduces **seek time** and speeds up file access.

Why it does not meaningfully help an SSD: an SSD has **no moving read/write head** and no spinning platter — data is stored electronically in flash memory cells and can be accessed **directly** at (almost) the same speed regardless of its physical location, so there is no "seek time" penalty for fragmented data to remove. Worse, flash memory cells can only be written/erased a **limited number of times** before they wear out; defragmenting an SSD involves large numbers of unnecessary extra write operations, which needlessly **shortens the drive's usable lifespan** for essentially no speed gain.

(!) Lỗi thường gặp: "Defragmenting" an SSD is not simply pointless — it is actively counterproductive, because it adds wear (extra write cycles) to a component with a finite number of write cycles, in exchange for a speed benefit that does not exist on that type of drive.

4.5 High-level and low-level languages

Low-level languages are tied closely to one specific processor's hardware. **Machine code** consists of the raw binary instructions the CPU executes directly. **Assembly language** represents those same instructions using short, human-readable **mnemonics** (e.g. `LDA`, `ADD`, `STA`), with each mnemonic corresponding one-to-one to a single machine code instruction.

High-level languages (e.g. Python, Java, C#) use syntax much closer to human/mathematical language. A single high-level statement typically expands into *many* machine code instructions once translated, which is what makes high-level languages quicker to write, easier to read, and largely portable across different types of hardware.

Feature	High-level language	Low-level language
Portability	Largely portable — the same source code can (with little or no change) be recompiled/reinterpreted to run on different processor types.	Not portable — machine code/assembly is written for one specific processor's instruction set and will not run unchanged on a different architecture.
Speed to write	Faster to write — syntax resembles English/mathematics; the programmer works with variables, loops and functions rather than individual registers.	Slower and more tedious — the programmer must manage individual registers and memory addresses explicitly.
Ease of debugging	Easier — readable syntax and descriptive variable/function names make errors easier to trace.	Harder — the programmer must reason in terms of raw registers, addresses and bit patterns.
Execution speed	Typically slower once translated (extra overhead; not hand-tuned to specific hardware).	Typically faster — instructions map directly and efficiently onto exactly what the processor executes.
Hardware-specificity	Hardware-independent — one language works across many processor families.	Hardware-specific — tied to one processor's particular instruction set.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ngôn ngữ bậc thấp (mã máy, hợp ngữ) gắn chặt với **một loại phần cứng cụ thể**: khó viết, khó sửa lỗi, nhưng chạy **nhanh** vì khớp trực tiếp với phần cứng. **Ngôn ngữ bậc cao** gần với ngôn ngữ con người: **di động** (portable) giữa các loại phần cứng khác nhau, viết nhanh và dễ sửa lỗi hơn nhiều, nhưng thường chạy **chậm hơn** sau khi dịch vì không tối ưu riêng cho từng loại CPU.

Ví dụ mẫu · Worked example

A robotics company writes firmware that must run as fast as possible on one specific embedded microcontroller and will never be reused on any other chip. A separate team writes a cross-platform mobile app that must run on many different phone models. Recommend a language category for each, with justification.

Firmware: a **low-level language** (assembly, or even hand-tuned machine code) is appropriate — execution speed is the priority, the target hardware is fixed and known, and portability is not required.

Mobile app: a **high-level language** is appropriate — the app must run, largely unchanged, across many different underlying hardware platforms, and development speed/maintainability matters more than squeezing out the last unit of execution speed.

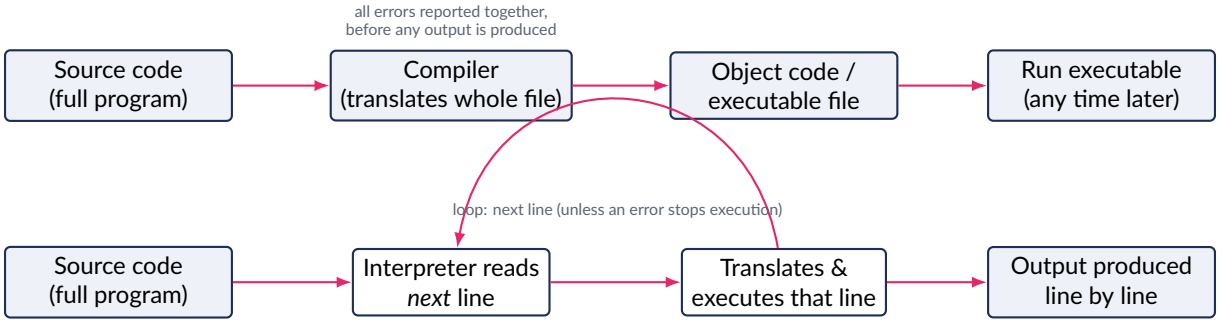
4.6 Translators

Because the CPU can only execute machine code, any program not already written in machine code must be converted by a **translator** before (or as) it runs.

Assembler: converts assembly language source code into machine code, on a (largely) one-to-one, instruction-by-instruction basis.

Compiler: translates the *entire* high-level source code into machine code (object code) *before* the program is run, producing a standalone executable file. All syntax errors found during compilation are reported together at the end; once compiled, the program runs quickly (no translation delay while it is running), but because the executable no longer resembles the original source, step-by-step debugging of the running program is harder.

Interpreter: translates and executes high-level source code *line by line* as the program runs, without producing a saved, standalone object-code file. It stops immediately at the *first* error it meets, which makes it excellent for testing and debugging small changes quickly, but the program runs more slowly overall, since every line must be re-translated every single time the program is run.



Top (compiler): whole program translated first into a standalone executable, then run — fast at run time, all errors reported together. **Bottom (interpreter):** each line is translated and executed immediately, one at a time, with no saved executable — stops at the first error met.

Ví dụ mẫu · Worked example

A program has a syntax error on line 50 of 100 lines. Contrast what happens when it is (a) compiled and (b) interpreted.

(a) **Compiler:** the whole file is scanned; the error on line 50 (and any others found) is reported, and *no* executable object code is produced until every error is fixed.

(b) **Interpreter:** lines 1–49 execute normally, one at a time; execution stops the instant line 50 is reached, reporting only that error — lines 51–100 are never even translated on this run.

Aspect	Compiler	Interpreter	Assembler
Translation timing	Whole program translated *before* it is run.	Each line translated *as* the program is run.	Whole assembly source translated before it is run.
Error reporting	All errors listed together, after compiling.	Execution stops at the *first* error found.	Errors listed after assembling.
Output produced	Standalone object code/executable file, kept and reused.	No saved executable — source re-translated every run.	Object code (machine code) file.
Run speed	Fast — no translation overhead at run time.	Slower overall — retranslated every time it runs.	Fast — already machine code.
Typical use	Distributing finished, performance-critical software.	Development, testing, scripting — rapid iterative feedback.	Low-level/embedded programming needing direct hardware control.

VN GIẢI THÍCH TIẾNG VIỆT

Hợp dịch (assembler): dịch hợp ngữ sang mã máy. **Biên dịch (compiler)**: dịch **toàn bộ** chương trình bậc cao sang mã máy **trước khi chạy**, tạo file thực thi độc lập, báo lỗi cùng lúc, chạy nhanh sau đó nhưng khó gỡ lỗi từng dòng. **Thông dịch (interpreter)**: dịch và chạy **từng dòng một** ngay khi chạy, không tạo file thực thi riêng, dừng ngay khi gặp lỗi đầu tiên — tiện gỡ lỗi nhưng chạy chậm hơn vì phải dịch lại mỗi lần chạy.

4.7 Integrated Development Environments (IDEs)

An **Integrated Development Environment (IDE)** bundles together, in one application, all the tools a programmer needs to write, translate, run and debug a program, instead of requiring separate standalone tools.

Khái niệm cốt lõi

A typical IDE provides:

- a **source code editor** with **syntax highlighting** (colour-coding keywords, variables and strings to make code easier to read), **auto-indentation** (automatically aligning nested code blocks) and **auto-completion/auto-correct** (suggesting or fixing likely code as the programmer types);
- **error diagnostics** that flag syntax errors *as the programmer types*, before the program is even run;
- a built-in **run-time environment** (a translator plus the facility to run/test the program without leaving the IDE);
- **debugging tools** for finding and fixing logic errors in a program that runs but produces the wrong result.

VN GIẢI THÍCH TIẾNG VIỆT

IDE (môi trường phát triển tích hợp) gộp chung mọi công cụ lập trình vào một phần mềm: **trình soạn thảo mã** (tô màu cú pháp, tự thụt lề, tự hoàn thiện/tự sửa mã), **chẩn đoán lỗi** (báo lỗi cú pháp ngay khi gõ, trước khi chạy), **môi trường chạy thử** tích hợp sẵn, và các **công cụ gỡ lỗi** để tìm lỗi logic trong chương trình chạy được nhưng cho kết quả sai.

Ví dụ mẫu · Worked example

A student typing a program forgets to close a bracket on the line they are currently writing. Explain what an IDE's error diagnostics feature would do, and why this is different from what happens with a plain text editor and a separate compiler.

As the student types, the IDE's editor continuously checks the code typed so far against the rules of the programming language's syntax. The instant the missing closing bracket makes the line (or the surrounding block) invalid, the IDE typically underlines or highlights the affected code and displays a message describing the problem (e.g. "expected ')' ") — **immediately**, while the student is still writing, before the program is ever run or compiled.

With a plain text editor and a separate compiler, no such checking happens while typing at all: the student would only discover the missing bracket *after* saving the file and running the compiler separately, by which point they may have written many more lines and lost track of exactly where the mistake was introduced. Real-time error diagnostics in an IDE therefore catch simple syntax mistakes far earlier and more precisely than a compile-time-only error report.

4.7.1 Debugging tools

Tool	What it lets the programmer do	Typical use
Breakpoint	Marks a chosen line so that, when the program is run in debug mode, execution automatically **pauses** exactly at that line, before it executes.	Pausing just before a suspicious calculation to inspect the program's state at that exact moment.
Single stepping (step into / step over)	Executes the program **one instruction/line at a time** under manual control after a pause. *Step into* follows execution *inside* a called subroutine/function; *step over* runs a called subroutine in one go without tracing inside it.	Watching exactly how each line changes a variable's value, one line at a time, to find precisely where a value first goes wrong.
Variable watch / report window	Displays the current values of chosen variables while the program is paused, updating live as the programmer steps through the code.	Confirming exactly *when*, during execution, a variable's value stops matching what was expected.

Ví dụ mẫu · Worked example

A student's program is supposed to add the numbers 1 to 5 and print the total, but it prints 10 instead of the expected 15. Explain how the student could use a **breakpoint** and a **variable watch window** to find the error.

The student sets a **breakpoint** on the line inside the loop that adds the current number to the running total, then adds both the loop counter (`i`) and the running total (`total`) to the **variable watch window**. Running the program in debug mode, execution pauses at the breakpoint on the first pass; the student then **single steps**, observing the watch window after each iteration: `i=1`: total becomes 1 (correct so far) → `i=2`: total becomes 3 → `i=3`: total becomes 6 → `i=4`: total becomes 10 → the loop then **exits without a fifth iteration**, and the program prints 10.

Because the watch window shows the total stopping at 10 after `i=4` rather than continuing to add 5, the student can see the loop's exit condition is checked *one iteration too early* (e.g. the loop runs `while i < 5` instead of `while i <= 5`), causing the final addition of 5 to be skipped. The breakpoint and watch window together let the student see *exactly which* iteration the loop

wrongly omits, without needing to guess from the final output alone.

(!) Lỗi thường gặp: A breakpoint only pauses execution — it does not by itself reveal the bug. The programmer must also inspect variable values (via a watch window) and step through the following lines to see where the actual behaviour first diverges from what was expected.

4.8 Practice

Bài tập luyện tập

Which of the following is an example of **application software** rather than system software?

(A) A disk defragmentation utility
(B) An assembler
(C) A spreadsheet program
(D) The operating system's file manager

Lời giải chi tiết

A spreadsheet program is run directly by a user to perform a specific task (calculations, budgeting) — this is **application software**. The other three options all manage or support the computer's own operation and are classed as system software. **(C)**.

Bài tập luyện tập

Explain why a compiler is classified as system software even though a programmer chooses to run it directly.

Lời giải chi tiết

System software is classified by *what it does for the computer*, not by whether a user consciously launches it. A compiler's purpose is to translate another program's source code into a form the processor can execute — it supports and enables *other* software to run, rather than performing an end-task such as writing a letter or editing a photo. Because its role is to manage/support the platform (translating programs so they can be executed), it is system software, just like the operating system or a utility program.

Bài tập luyện tập

Which OS function is responsible for using device drivers to translate a generic "print" command into instructions a specific printer model understands?

(A) Memory management
(B) Peripheral/device management
(C) File management
(D) Multitasking

Lời giải chi tiết

Device drivers, managed by the OS's **peripheral/device management** function, translate general commands into hardware-specific instructions for each connected device. **(B)**.

Bài tập luyện tập

Three applications are open at once on a single-core processor: a browser, a spreadsheet, and a music player. Explain the role of the **scheduler** in making this feel like true simultaneous operation.

Lời giải chi tiết

A single processor core can only truly execute one instruction stream at a time. The OS's **scheduler** repeatedly decides which of the three waiting programs is given the CPU next, and for how long (a short time-slice), according to a scheduling policy. Because these time-slices are extremely short and switching happens very rapidly, the user perceives the browser, spreadsheet and music player as running **simultaneously**, even though, at any single instant, only one of them is actually executing on the CPU. This rapid, scheduler-driven switching is what produces the effect of **multitasking**.

Bài tập luyện tập

A word-processing program has 8 GB of data open across several documents, but the computer only has 4 GB of physical RAM installed. Which OS function makes it possible to keep working, and how does it work?

Lời giải chi tiết

Memory management, specifically **virtual memory**, makes this possible. The OS temporarily moves data that is not currently needed from RAM onto secondary storage (e.g. the hard disk/SSD), freeing physical RAM for what is needed right now; when that swapped-out data is needed again, the OS brings it back into RAM (swapping out something else if necessary). This lets the computer behave as though it has more RAM than is physically installed, at the cost of some extra delay when data must be swapped in from the (much slower) secondary storage.

Bài tập luyện tập

A printer interrupt (priority 2) occurs while the CPU is already handling a disk-error interrupt (priority 1, where 1 is the highest priority). What happens?

(A) The printer interrupt is handled first, pausing the disk-error ISR

(B) The disk-error ISR continues uninterrupted, since it already has higher priority

(C) Both are ignored until the current program finishes

(D) The CPU crashes

Lời giải chi tiết

Interrupt priorities mean a **lower-priority** interrupt (here, the printer, priority 2) cannot interrupt the handling of a **higher-priority** one (the disk error, priority 1) already in progress. The disk-error ISR continues; the printer interrupt is queued until it finishes. **(B)**.

Bài tập luyện tập

A keyboard interrupt (priority 3, lowest) begins being serviced. While its ISR is running, a disk I/O interrupt (priority 2) arrives, and while *that* ISR is running, a timer interrupt (priority 1,

highest) arrives. List the order in which the four things — keyboard ISR, disk ISR, timer ISR, and the original program — *finish/resume*, and explain the underlying rule.

Lời giải chi tiết

Finishing/resuming order: timer ISR completes first → disk ISR resumes and completes → keyboard ISR resumes and completes → original program resumes.

Rule: each new, higher-priority interrupt preempts (pauses) whatever is currently running, saving its state on the stack before jumping to the new ISR. Because each pause is nested on top of the previous one, the *completion* order is exactly the **reverse** of the *interruption* order: the last thing paused (nothing — the timer ISR runs uninterrupted since priority 1 is highest) finishes first, and the very first thing paused (the original program) resumes last, only after every ISR stacked above it has completed.

Bài tập luyện tập

Put the following steps of interrupt handling into the correct order: (i) CPU jumps to the Interrupt Service Routine (ISR), (ii) an interrupt signal is received, (iii) CPU finishes the current fetch–execute cycle, (iv) CPU restores the saved register state and resumes the original program, (v) CPU saves the current register state onto the stack.

Lời giải chi tiết

Correct order: **(ii)** interrupt signal received → **(iii)** CPU finishes the current fetch–execute cycle → **(v)** register state saved onto the stack → **(i)** CPU jumps to the ISR → **(iv)** once the ISR completes, saved state is restored and the original program resumes.

Bài tập luyện tập

Which of the following best describes what antivirus software does?

(A) Reorganises fragmented files on a hard disk for faster access

(B) Scans files/programs against known virus signatures and quarantines or deletes infected files

(C) Reduces file size for storage or transmission

(D) Automatically copies files to a separate location on a schedule

Lời giải chi tiết

Antivirus software compares files and running programs against a database of known virus signatures (and uses heuristics for unknown threats), quarantining or deleting anything infected. **(B)**. (The other options describe disk defragmentation, file compression, and backup software respectively.)

Bài tập luyện tập

A technician runs a disk defragmentation utility on an old magnetic HDD that has become sluggish, and separately considers running the same utility on a new SSD-based laptop that also feels slow. Explain the likely outcome in each case.

Lời giải chi tiết

HDD: over time, files have likely become split into fragments stored in scattered, non-contiguous sectors as the disk filled and files were resized/deleted. Defragmenting rewrites the disk so each file's data is stored contiguously, reducing how far the mechanical read/write head must physically travel (**seek time**) to read a file — this is likely to noticeably speed up file access.

SSD: an SSD has no moving read/write head; any data block can be accessed electronically in roughly the same time regardless of its physical location, so there is no seek-time penalty for fragmentation to remove. Defragmenting the SSD would therefore give little or no speed improvement, while adding many unnecessary write operations that wear out the SSD's flash memory cells (which have a limited number of write cycles) — the laptop's slowness has some other cause (e.g. too little free space, too many background programs, insufficient RAM) that defragmentation will not fix.

Bài tập luyện tập

A company wants to protect itself against losing important files if a hard drive fails, and separately wants to reduce the size of large files before emailing them. Identify the correct utility for each need.

(A) Backup software for both needs

(B) Backup software for drive failure protection; file compression for reducing email attachment size

(C) File compression for drive failure protection; backup software for reducing attachment size

(D) Antivirus software for both needs

Lời giải chi tiết

Backup software automatically copies files to a separate location/medium, so they can be restored if the original drive fails — this protects against data loss. **File compression** reduces file size, making large attachments faster and more practical to email. These are two different tools for two different problems. **(B)**.

Bài tập luyện tập

Classify each as high-level or low-level, and justify one classification: (a) Python, (b) assembly language using mnemonics like `MOV` and `ADD`.

Lời giải chi tiết

(a) Python is a **high-level language** — its syntax resembles human/mathematical language, is portable across different hardware, and each statement expands into many machine instructions.
(b) Assembly language is a **low-level language** — each mnemonic (`MOV`, `ADD`) corresponds directly, one-to-one, to a single machine code instruction, and it is written for one specific processor's instruction set.

Bài tập luyện tập

Give *one* advantage of writing a program in a high-level language rather than assembly language, and *one* advantage of assembly language over a high-level language for a specific use case.

Lời giải chi tiết

High-level advantage: the source code is **portable** — the same program can be translated to run on different processor types with little or no modification, whereas assembly must be rewritten for each different instruction set. (Other valid advantages: quicker to write; easier to debug, due to readable, English-like syntax.)

Assembly advantage: for time-critical embedded code (e.g. controlling precise hardware timing), assembly lets the programmer control exactly which registers and instructions are used, producing code that runs **faster and more predictably** than a high-level language translated by a general-purpose compiler, because there is no translation overhead or generic, non-hardware-specific code generation to work around.

Bài tập luyện tập

Which statement correctly distinguishes a compiler from an interpreter?

(A) A compiler translates and executes one line at a time; an interpreter translates the whole program before running it

(B) A compiler translates the whole program before it runs and produces a standalone executable; an interpreter translates and executes line by line as the program runs

(C) A compiler and an interpreter both stop at the first error found

(D) An interpreter always runs a program faster than a compiled version of the same program

Lời giải chi tiết

A compiler translates the entire source program before execution, producing a standalone executable/object code file. An interpreter translates and executes the source code line by line, as the program runs, without saving a standalone executable. **(B)**.

Bài tập luyện tập

A program has a syntax error on line 50 of 100 lines. Contrast what happens when it is (a) compiled and (b) interpreted.

Lời giải chi tiết

(a) **Compiler:** the entire file is scanned during compilation; the error on line 50 (and any other errors present) is reported all together, and no executable object code is produced until every reported error is fixed.
(b) **Interpreter:** lines 1–49 are translated and executed normally, one at a time; execution halts the instant line 50 is reached, reporting only that single error — lines 51–100 are never even translated on this run, since the interpreter never gets that far.

Bài tập luyện tập

A student is developing and frequently testing small snippets of a new program, running each change immediately to see the result. Which type of translator suits this workflow best, and why?

(A) Compiler, because it reports all errors at once

(B) Interpreter, because it translates and runs code line by line with immediate feedback

(C) Assembler, because it works with mnemonics

(D) No translator is needed for testing

Lời giải chi tiết

An interpreter translates and executes code line by line, so the student gets **immediate feedback** on each small change without waiting for a full compilation — ideal for rapid, iterative testing. **(B)**.

Bài tập luyện tập

A software company is about to release a finished commercial video game to customers, where fast run-time performance matters and the source code should not be exposed. Recommend a translator type and justify your answer.

Lời giải chi tiết

A **compiler** is recommended. Compiling the entire program in advance produces a standalone executable file that runs quickly (there is no translation overhead while the customer is playing the game), and the distributed file is machine code rather than readable source code, so the original program logic is not exposed to customers. An interpreter would be unsuitable here, since it would need to re-translate the source every time the game runs (slower) and typically requires the readable source code to be present on the customer's machine.

Bài tập luyện tập

Which of the following is *not* typically a feature provided by a source code editor within an IDE?

(A) Syntax highlighting

(B) Auto-indentation

(C) Automatically fixing all logic errors in the program without programmer input

(D) Auto-completion of code as it is typed

Lời giải chi tiết

Syntax highlighting, auto-indentation and auto-completion are all standard IDE editor features that assist with *writing* code. However, an IDE cannot automatically detect and fix **logic errors** (the program runs but produces the wrong result) without programmer input — diagnosing and fixing logic errors requires the programmer to use debugging tools (breakpoints, stepping, variable watches) and their own understanding of what the program *should* do. **(C)**.

Bài tập luyện tập

Explain the difference between **step into** and **step over** when single-stepping through a program in an IDE debugger, using a program that calls a subroutine named `calculateTax()`.

Lời giải chi tiết

When execution is paused on the line that calls `calculateTax()`: **step into** makes the debugger follow execution *inside* the `calculateTax()` subroutine, pausing again on its very first line, so the programmer can trace what happens line by line *within* the subroutine itself. **Step over** instead runs the entire `calculateTax()` subroutine in one go (without pausing inside it) and pauses again on the line immediately *after* the call, in the calling code — useful when the programmer trusts that subroutine works correctly and only wants to continue tracing the surrounding code.

Bài tập luyện tập

A student's loop is meant to add the numbers 1 to 5 and print the total (expected: 15), but it prints 10. Using a breakpoint at the addition line and a variable watch window showing `i` and `total`, the student observes: `i=1, total=1`; `i=2, total=3`; `i=3, total=6`; `i=4, total=10`; then the loop exits. Explain what the bug is and how the watch window revealed it.

Lời giải chi tiết

The watch window shows the running total correctly increasing after each of the first four iterations ($1, 3, 6, 10$), matching $1 + 2 + 3 + 4$. But the loop then exits *without* a fifth iteration that would add 5 (which would give the expected total of 15). This tells the student the loop's **exit condition is checked one iteration too early** — for example, the loop is written as `while i < 5` (which lets `i` run only through $1, 2, 3, 4$) instead of `while i <= 5` (or equivalent, so that `i=5` is also included). Watching `i` and `total` update live, step by step, let the student see *exactly which* iteration was wrongly skipped, rather than only knowing the *final* answer was wrong.

Bài tập luyện tập

Which debugging tool would be most useful for pausing a program's execution at a specific, chosen line so its state can be examined at that exact moment?

(A) Syntax highlighting
(B) A breakpoint
(C) File compression
(D) A checksum

Lời giải chi tiết

A **breakpoint** marks a chosen line so that, when the program is run in debug mode, execution automatically pauses there, allowing the programmer to inspect the program's state at that precise point. **(B)**.

Bài tập luyện tập

Explain why an IDE's error diagnostics (which flag syntax errors as the programmer types) do not remove the need for the debugging tools (breakpoints, single stepping, variable watch) described in this chapter.

Lời giải chi tiết

Error diagnostics can only detect **syntax errors** — code that breaks the rules of the programming language itself (e.g. a missing bracket or misspelt keyword) — because these can be checked automatically against the language's grammar as the programmer types. They cannot

detect **logic errors**, where the code is syntactically valid and the program runs to completion, but produces the wrong result because of a flaw in the programmer's reasoning (e.g. an incorrect loop condition, as seen earlier in this chapter). Finding a logic error requires a human to compare the program's *actual* behaviour, observed step by step via breakpoints, single stepping and a variable watch window, against its *intended* behaviour — something no automated syntax check can do.

Bài tập luyện tập

A computer is running a background file-transfer program (low priority) when a timer interrupt (priority 1, highest in this system) and a mouse-click interrupt (priority 4, lowest) both arrive while the file-transfer program is executing, with the timer interrupt arriving slightly before the mouse-click interrupt. Which is serviced first, and why?

(A) The mouse-click interrupt, because input devices always take priority

(B) The timer interrupt, because it has the higher priority (lower priority number) regardless of arrival order

(C) Whichever arrived first is always serviced first, regardless of priority

(D) Neither is serviced until the file transfer completes

Lời giải chi tiết

The **timer interrupt** is serviced first. Interrupt priority — not arrival order — determines service order whenever more than one interrupt is waiting: priority 1 (the timer) outranks priority 4 (the mouse click), so even though both arrived while the file transfer was running, the timer interrupt is dealt with first, and the mouse-click interrupt waits until the timer's ISR has completed (or, if the file-transfer program resumes first, until it is next interrupted). **(B)**.

Bài tập luyện tập

A programmer wants to check the exact value of a loop counter and an accumulator variable at several points while a program runs, without modifying the program's own code to add temporary print statements. Which IDE debugging feature is designed exactly for this, and how does it help compared to inserting temporary print statements?

Lời giải chi tiết

The **variable watch/report window** is designed for this. The programmer selects the variables of interest (e.g. the loop counter and the accumulator) once, and the IDE displays their current values continuously, updating automatically every time execution pauses (e.g. at a breakpoint or after a single step) — without the programmer needing to edit, add, or later remove any lines of the program's own source code. This is more convenient and less error-prone than inserting temporary print statements, which must be manually written for each variable, placed at exactly the right lines, and then remembered and removed afterwards so they do not remain in the finished program.

Bài tập luyện tập

A company's IT policy requires (i) every user's files to be recoverable if a hard drive fails, (ii) all incoming email attachments to be checked for malicious code before opening, and (iii) large design files to take up as little storage space as possible. State which type of utility software

satisfies each requirement.

Lời giải chi tiết

(i) **Backup software** — it automatically copies files to a separate storage location/medium on a schedule, so they can be restored if the original drive fails.
(ii) **Antivirus software** — it scans files (including email attachments) against known virus signatures and heuristics, and quarantines or deletes anything infected before it can cause harm.
(iii) **File compression software** — it reduces the size of large files (typically without losing any data), reducing the storage space they occupy.

Bài tập luyện tập

A server has five waiting processes of equal importance. Which scheduling policy would treat them most fairly, and what is one drawback of using that same policy in a system where some tasks are genuinely more urgent than others?

(A) Priority-based scheduling; drawback: urgent tasks never run

(B) Round-robin scheduling; drawback: it cannot distinguish an urgent task from a routine one

(C) Round-robin scheduling; drawback: it always favours the first process forever

(D) Priority-based scheduling; drawback: it gives every process exactly equal CPU time

Lời giải chi tiết

Round-robin scheduling treats equally-important waiting processes most fairly, since it gives every process an equal, fixed-length time-slice in turn. Its drawback is that it has no concept of urgency: if some tasks genuinely need faster attention than others, round-robin still gives them no more CPU time than any routine task, potentially delaying an urgent task unnecessarily. **(B)**.

Bài tập luyện tập

A calculator program attempts to divide a number by zero during execution, which is not a valid mathematical operation the processor can complete. Explain why this situation is described as a **software interrupt**, and outline how the OS is likely to respond.

Lời giải chi tiết

This is a **software interrupt** because the signal to the processor is generated by the currently running program's own instruction (attempting an invalid operation) rather than by an external hardware device such as a keyboard or disk controller. When the divide-by-zero condition is detected, the CPU raises an interrupt in exactly the same general way as a hardware interrupt: it finishes what it can of the current cycle, saves the program's state, and jumps to an appropriate **Interrupt Service Routine**. Because there is no mathematically valid result to continue with, the OS's ISR for this condition typically reports a run-time error to the user (or to the program, so it can handle the error itself) rather than resuming normal execution at the point of the failed division.

The Internet and Its Uses

Mục tiêu Unit: Hạ tầng Internet (ISP, IP address, DNS, hosting) và sự khác biệt giữa Internet và World Wide Web; cấu trúc URL; trình duyệt web và cookie (session/persistent); các hình thức tấn công an ninh mạng (virus, worm, trojan, spyware, ransomware, phishing, pharming, brute-force, data interception, SQL injection, DoS); và các biện pháp bảo vệ (SSL/TLS, firewall, anti-malware, mật khẩu mạnh, xác thực hai yếu tố, proxy server, sinh trắc học, chứng chỉ số).

5.1 Internet infrastructure

The **Internet** is a vast, physical network of interconnected computers, servers and communication links (cables, satellite links, wireless connections) spanning the entire globe. It exists independently of any particular service that runs on it — it is the underlying *infrastructure*, not any one application.

Khái niệm cốt lõi

An ordinary user does not connect directly into this global network. Instead, a user's device connects through an **Internet Service Provider (ISP)** — a company that provides an individual or organisation with a connection to the Internet (and typically also assigns the user's device an IP address, provides a DNS lookup service, and offers email/hosting services). Once connected via an ISP, a device can exchange data with any other connected device anywhere in the world, subject to the rules of the underlying communication protocols.

VN **GIẢI THÍCH TIẾNG VIỆT**

Internet là mạng lưới vật lý toàn cầu gồm hàng triệu máy tính, máy chủ và đường truyền (cáp, vệ tinh, không dây) kết nối với nhau — đây là **hạ tầng vật lý**, không phải một dịch vụ cụ thể nào. Người dùng thông thường không kết nối trực tiếp vào mạng lưới này mà thông qua một **ISP (Nhà cung cấp dịch vụ Internet)** — công ty cung cấp đường kết nối Internet, thường đi kèm cấp địa chỉ IP, dịch vụ tra cứu DNS, và dịch vụ email/hosting.

5.1.1 IP addresses

Every device connected to a network needs a way to be uniquely identified so that data sent across the network reaches the correct destination. This identifier is the device's **IP address**.

An **IP address** is a numerical address that identifies a specific device on a network. The most familiar form (IPv4) is written as four numbers between 0 and 255 separated by dots, e.g. `192.168.1.10`. Every device that sends or receives data directly over the Internet — a home router, a web server, a smartphone — must have an IP address, in exactly the same way every house on a street needs a unique street address before the postal service can deliver a letter to it.

VN **GIẢI THÍCH TIẾNG VIỆT**

Địa chỉ IP là một địa chỉ dạng số dùng để nhận diện một thiết bị cụ thể trên mạng — ví dụ `192.168.1.10`. Mọi thiết bị gửi/nhận dữ liệu trực tiếp qua Internet đều cần có địa chỉ IP, giống như mỗi ngôi nhà cần có địa chỉ bưu điện riêng để thư từ được giao đến đúng nơi.

5.1.2 Hosting

A website's files — its HTML pages, images, videos, stylesheets and scripts — have to physically live somewhere, on a computer that is permanently connected to the Internet and ready to send those files out whenever a visitor requests them. This is the role of **hosting**.

Khái niệm cốt lõi

A **web server** is a computer (often one of many working together, run by a hosting company) that stores a website's files and serves (sends) them to a visitor's browser whenever that browser requests them. **Hosting** is the service of storing and maintaining these website files on a web server that is permanently online, so the website is available to visitors at any time, from anywhere in the world.

VN **GIẢI THÍCH TIẾNG VIỆT**

Máy chủ web (web server) là máy tính lưu trữ toàn bộ file của một website (HTML, hình ảnh, video, mã) và gửi các file này đến trình duyệt của người truy cập khi có yêu cầu. **Hosting** là dịch vụ lưu trữ và duy trì các file này trên một máy chủ luôn hoạt động (online 24/7), để website luôn sẵn sàng phục vụ người dùng ở bất kỳ đâu, bất kỳ lúc nào.

5.1.3 The Internet is not the World Wide Web

(!) Lỗi thường gặp: The terms "the Internet" and "the Web" are very often used interchangeably in everyday speech, but in Cambridge IGCSE Computer Science they name two genuinely different things, and exam questions frequently test whether a candidate can tell them apart.

Khái niệm cốt lõi

The **Internet** is the physical, global network of interconnected devices and communication links described above — it can carry *any* kind of data, for *any* service (email, file transfer, video calls, online gaming, and the Web).

The **World Wide Web (WWW)** is just *one* of the many services that runs *on top of* the Internet: a huge, interlinked collection of webpages, joined together by **hyperlinks**, that a user accesses using the **HTTP** or **HTTPS** protocol through a web browser. Email, for example, also uses the Internet but is *not* part of the Web — it uses entirely different protocols (such as SMTP and IMAP), not HTTP/HTTPS and not webpages/hyperlinks.

VN **GIẢI THÍCH TIẾNG VIỆT**

Internet là hạ tầng mạng vật lý toàn cầu, có thể mang **bất kỳ loại dữ liệu** nào cho nhiều dịch vụ khác nhau (email, gọi video, chơi game trực tuyến, và cả Web). **World Wide Web (WWW)** chỉ là **một trong nhiều dịch vụ** chạy *trên nền* Internet: một tập hợp khổng lồ các trang web liên kết với nhau qua **siêu liên kết (hyperlink)**, truy cập bằng giao thức **HTTP/HTTPS** thông qua trình duyệt. Email cũng dùng Internet nhưng **không phải** một phần của Web, vì nó dùng giao thức khác hẳn (SMTP, IMAP), không phải HTTP/HTTPS.

Ví dụ mẫu · Worked example

State, with a reason, whether each of the following uses the World Wide Web: (a) reading a news article at a website in a browser; (b) sending an email using a mail client; (c) using an online banking website to view a statement.

(a) **Yes** — a news article is a webpage, retrieved via HTTP/HTTPS and viewed in a browser: this is exactly what the Web is.
(b) **No** — email uses the Internet (the underlying physical network) to travel, but it is transmitted using mail protocols such as SMTP/IMAP, not HTTP/HTTPS, and is not a webpage linked by hyperlinks.
(c) **Yes** — an online banking statement is delivered as a webpage over HTTPS, so it is part of the Web (as well as, more generally, using the Internet).

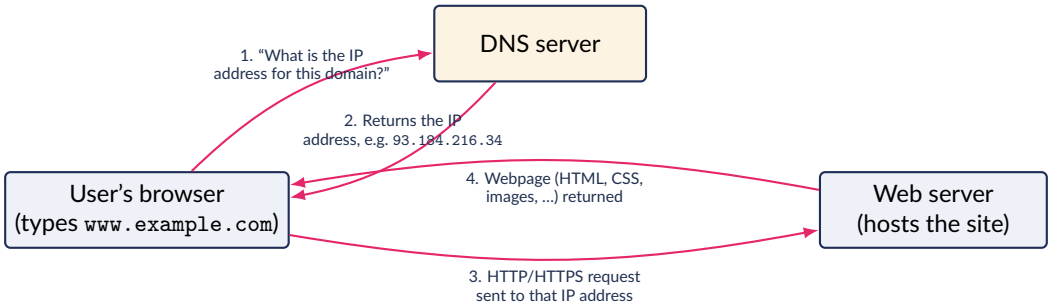
5.1.4 DNS: translating domain names into IP addresses

Every website is really hosted at a numerical IP address, but almost no one types an IP address into a browser — instead, people type a memorable **domain name** such as `www.example.com`. Something has to convert that human-friendly name into the numerical IP address the network actually needs, and that job belongs to the **Domain Name System (DNS)**.

DNS (Domain Name System) translates a human-readable domain name into the numerical IP address of the web server that hosts it. A **DNS server** holds (or can look up) a database of domain-name-to-IP-address mappings, so that a browser can find the correct server to contact for any domain name a user types in.

VN **GIẢI THÍCH TIẾNG VIỆT**

DNS (Hệ thống tên miền) chuyển đổi một tên miền dễ nhớ (như `www.example.com`) thành địa chỉ IP số của máy chủ lưu trữ website đó. **Máy chủ DNS** lưu (hoặc tra cứu được) một cơ sở dữ liệu ánh xạ tên miền sang địa chỉ IP, để trình duyệt tìm được đúng máy chủ cần kết nối tới ứng với tên miền người dùng gõ vào.



Resolving a domain name and retrieving a page: (1)–(2) the browser asks a DNS server to resolve the typed domain name into an IP address; (3)–(4) only once it has that IP address does the browser send an HTTP/HTTPS request to the correct web server and receive the page back.

Ví dụ mẫu · Worked example

A user types `www.example.com` into their browser's address bar for the very first time. Explain, in order, what must happen before the page appears on screen.

1. The browser sends the domain name `www.example.com` to a **DNS server**, asking for the corresponding IP address.

2. The DNS server looks up (or forwards the request further until it finds) the IP address associated with that domain name, and returns it to the browser — e.g. `93.184.216.34`.
3. The browser uses this IP address to send an **HTTP** (or **HTTPS**) request to the correct **web server**, asking for the specific page.
4. The web server sends back the requested page's files (HTML, CSS, images, etc.), and the browser **renders** them into the visual page the user sees.

Without step 1–2 (the DNS lookup), the browser would have no way of knowing which physical server on the entire Internet to even contact.

(!) Lỗi thường gặp: Do not describe DNS as "the Internet's address book that the user reads". The user never sees this lookup happen — it occurs automatically and near-instantly, behind the scenes, every time a domain name is used, before any request for a page is sent.

5.1.5 URL structure

The full text a user types (or clicks) to reach a specific webpage is called a **URL (Uniform Resource Locator)**. A URL is built from several distinct parts, each with its own job.

Ví dụ mẫu · Worked example

Break down the following URL into its component parts: `https://www.example.com/products/index.html`

Part	Value in this URL	Purpose
Protocol	`https://`	States which set of rules will be used to transfer the page; `https` additionally encrypts the data in transit (see later in this chapter).
Domain name	`www.example.com`	Identifies *which website/server* is being requested; this is exactly the part a DNS server translates into an IP address.
Path	`/products/`	Identifies the folder (directory) on the web server in which the requested file is stored.
Filename	`index.html`	Identifies the specific file (webpage) being requested from within that folder.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **URL** gồm: **giao thức** (`https://` — quy định cách truyền dữ liệu, có mã hoá hay không); **tên miền** (`www.example.com` — xác định website nào, chính là phần DNS dịch sang IP); **đường dẫn** (`/products/` — thư mục chứa file trên máy chủ); và **tên file** (`index.html` — file cụ thể được yêu cầu).

Ví dụ mẫu · Worked example

For the URL `https://www.igcse-cs.org/notes/chapter5.pdf`, identify the protocol, domain name, path, and filename.

Protocol: `https` (encrypted HTTP). **Domain name:** `www.igcse-cs.org`. **Path:** `/notes/`. **Filename:** `chapter5.pdf`.

5.2 Browsers and cookies

5.2.1 Functions of a web browser

A **web browser** (e.g. Chrome, Safari, Firefox, Edge) is the application software a user runs to access the Web: it is the program responsible for sending requests to web servers, receiving the raw files that come back, and turning them into something a person can actually read and interact with.

Khái niệm cốt lõi

A web browser:

- **renders HTML and CSS** — it reads the HTML markup (structure/content) and CSS (styling) sent back by a web server and displays them as a formatted, visual page, rather than showing the user the raw underlying code;
- performs the **DNS lookup** and sends the **HTTP/HTTPS request** needed to fetch a page, as described in the previous section;
- manages **bookmarks/favourites**, so a user can save and quickly return to a specific page;
- keeps a **history** of previously visited pages;
- manages multiple **tabs/windows**, allowing several pages to be open and used at once;
- stores and manages **cookies** on behalf of websites, as described below.

VN **GIẢI THÍCH TIẾNG VIỆT**

Trình duyệt web là phần mềm giúp người dùng truy cập Web: **hiển thị (render)** HTML/CSS thành trang có định dạng; thực hiện **tra cứu DNS** và gửi yêu cầu HTTP/HTTPS; quản lý **bookmark/favourite**, **lịch sử duyệt web**, nhiều **tab/cửa sổ** cùng lúc; và lưu trữ, quản lý **cookie** thay cho các website.

5.2.2 Cookies

A **cookie** is a small text file that a website asks the user's browser to store on the user's own device. The website can later ask the browser to send that same cookie back, allowing the site to "remember" something about that particular user between one request and the next.

Khái niệm cốt lõi

Cookies are commonly used to store: **login state** (so a user is not asked to log in again on every single page); **shopping cart contents** (which items a user has added, while they continue to browse); **preferences** (e.g. chosen language, display theme, currency); and **tracking data** (recording pages visited and behaviour, often used to build a profile of a user's interests for targeted advertising).

VN **GIẢI THÍCH TIẾNG VIỆT**

Cookie là một file văn bản nhỏ mà website yêu cầu trình duyệt lưu trên thiết bị người dùng, để website "ghi nhớ" thông tin của người dùng đó giữa các lần truy cập. Cookie thường lưu: **trạng thái đăng nhập**, **giỏ hàng**, **tuỳ chọn cá nhân** (ngôn ngữ, giao diện), và **dữ liệu theo dõi**

hành vi để phục vụ quảng cáo nhắm mục tiêu.

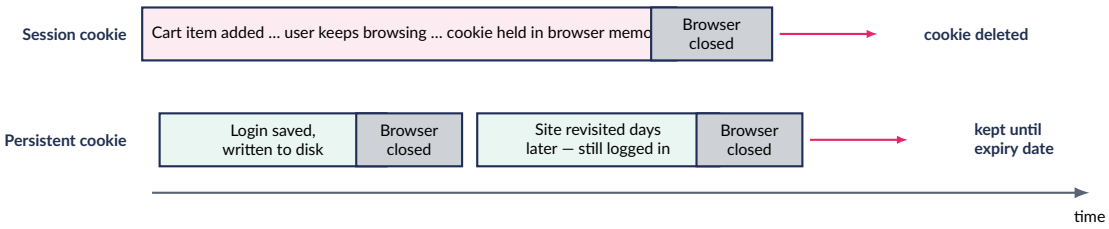
Not every cookie behaves the same way, however: cookies differ in *how long* they persist on the user's device, and this distinction matters a great deal in practice.

A **session cookie** is temporary: it is held only in the browser's memory for the duration of a single browsing session, and is **automatically deleted the moment the browser is closed**. It is typically used for data that only needs to exist while the user is actively using the site right now — most classically, the contents of a shopping cart while the user continues browsing an online store during one visit.

A **persistent cookie** is instead **saved to disk** with an expiry date, and survives being closed and reopened — it remains available the *next* time the user visits the site, potentially days, months or years later, until it expires or the user manually deletes it. It is typically used for data that should be remembered *across* visits, such as staying logged in automatically, or remembering a user's chosen language or theme the next time they return.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cookie phiên (session cookie): tạm thời, chỉ tồn tại trong bộ nhớ trình duyệt trong một phiên duyệt web, **tự động bị xoá khi đóng trình duyệt** — ví dụ điển hình là giỏ hàng mua sắm trong một lần ghé thăm. **Cookie bền vững (persistent cookie)**: được **lưu xuống ổ đĩa** kèm ngày hết hạn, vẫn còn sau khi đóng và mở lại trình duyệt, tồn tại đến **lần truy cập kế tiếp** — ví dụ tự động đăng nhập hoặc ghi nhớ ngôn ngữ/giao diện ưa thích cho lần sau.



A session cookie (top) does not survive the browser closing at all. A persistent cookie (bottom) survives being closed and reopened, and is only removed once its stored expiry date is reached (or the user deletes it manually).

Ví dụ mẫu · Worked example

A user shops on an online store: they add three items to their cart, then close the browser by accident before checking out. They reopen the browser an hour later and revisit the store. Explain what they are likely to find, and why.

If the cart was implemented using a **session cookie**, the cookie was deleted automatically the moment the browser closed, so the cart will be **empty** when the store is revisited — the site has no remaining record of what was previously added, since that data was never saved to disk. If instead the store used a **persistent cookie** to remember the cart (many real stores do this deliberately, precisely to avoid losing sales this way), the cookie would still be present on disk, unaffected by the browser closing, and the three items would **still be in the cart** when the store is revisited.

Ví dụ mẫu · Worked example

A news website remembers a returning visitor's chosen "dark mode" display setting every time they visit over the following year, without the user ever logging in. Identify the type of cookie responsible, and explain how you know.

This must be a **persistent cookie**. A session cookie would be deleted the instant the browser was closed after the setting was first chosen, so it could never still be present a year — or even a day — later. Because the preference survives being closed and reopened many times over a long period, it must have been **written to disk with an expiry date** far in the future, which is exactly what defines a persistent cookie.

(!) Lỗi thường gặp: "The website remembers me" is not, by itself, evidence of *which* type of cookie is in use — it depends entirely on whether that memory survives the browser being closed. If it does not survive closing the browser at all, it is a session cookie; if it survives being closed and reopened (even much later), it is a persistent cookie.

5.3 Cyber security threats

Because the Internet connects a user's device to millions of unknown other devices, it also creates opportunities for people to try to damage data, steal information, or disrupt services. This section describes the main categories of threat named in the syllabus.

5.3.1 Malware

Malware ("malicious software") is a general term for any software deliberately written to damage, disrupt or gain unauthorised access to a computer system. Different types of malware are distinguished chiefly by *how they spread* and *what they do* once installed.

Virus: attaches itself to a genuine file or program, and **requires that host file to be opened or run by a user** before it can activate and spread; once active, it can replicate itself into other files and corrupt or delete data.

Worm: is a **self-replicating** program that spreads automatically across a network from computer to computer, **without needing a host file and without requiring any user action** — it can often exploit a security weakness to copy itself onto other machines by itself, typically consuming large amounts of bandwidth and system resources as it spreads.

Trojan horse: disguises itself as legitimate, desirable software (e.g. a game or a free utility) so that a user is tricked into installing it themselves; it does **not** self-replicate, but once installed it can perform hidden, harmful actions — classically, opening a "back door" that lets an attacker access the system.

Spyware: installs itself and then **secretly monitors** a user's activity (e.g. keystrokes, browsing habits, screen contents) without their knowledge, transmitting the gathered information back to a third party — often used to steal passwords or financial details.

Ransomware: **encrypts** the victim's files (or otherwise locks them out of their own system) and then demands a payment (a "ransom", often in cryptocurrency) in exchange for the key or method needed to restore access.

VN **GIẢI THÍCH TIẾNG VIỆT**

Virus: gắn vào file/chương trình, cần người dùng **mở/chạy file đó** mới kích hoạt và lây lan. **Worm**: **tự nhân bản** và lây lan qua mạng **tự động**, không cần file chủ, không cần người dùng thao tác. **Trojan**: giả dạng phần mềm hợp pháp để lừa người dùng tự cài đặt, không tự nhân

bản, thường mở "cửa hậu" cho kẻ tấn công. **Spyware**: âm thầm theo dõi hoạt động người dùng (gõ phím, duyệt web) và gửi dữ liệu ra ngoài. **Ransomware**: mã hoá dữ liệu nạn nhân và đòi tiền chuộc để khôi phục quyền truy cập.

Ví dụ mẫu · Worked example

A user downloads what they believe is a free wallpaper app; after installing it, an attacker gains remote access to the computer through a hidden channel the app secretly opened. A separate incident, at a different company, sees a piece of malicious code spread by itself from one networked computer to another overnight, with no file ever being opened by any employee, saturating the office network connection. Identify the type of malware in each case.

First incident: a **trojan horse** — the malicious program disguised itself as something desirable (a wallpaper app) to trick the user into installing it themselves, then opened a hidden back door; it did not self-replicate.
Second incident: a **worm** — it spread automatically across the network, from machine to machine, without needing any host file to be opened or any user action at all, and its spreading activity consumed network bandwidth.

(!) Lỗi thường gặp: A common exam slip is confusing a **virus** with a **worm**. The test is always: does it need a host file opened by a user to activate and spread (virus), or does it spread by itself across a network with no user action required (worm)? "Self-replicating" on its own does not distinguish them — both can replicate; only the *requirement for a user to open a file* distinguishes a virus from a worm.

5.3.2 Phishing and pharming

Two further, distinctly named threats rely on a user ending up at a fraudulent destination or being persuaded to hand over information — but they achieve this in fundamentally different ways.

Phishing is a fraudulent attempt — typically an email, text message or other communication made to look as if it comes from a legitimate, trusted organisation (a bank, a delivery company, an employer) — that **tricks** the user into clicking a malicious link or voluntarily entering personal/financial details into a fake website or form. Phishing depends entirely on **deceiving the user** into taking an action.

Pharming instead relies on malicious code (or corruption of DNS records, or of the `hosts` file on the victim's own device) that **automatically redirects** a user to a fraudulent website, **even when the user types the correct, genuine URL themselves**. Pharming requires no deception of the user at all at the moment of the attack — the redirection happens silently and automatically, regardless of what the user clicks or types.

VN **GIẢI THÍCH TIẾNG VIỆT**

Phishing: gửi email/tin nhắn giả mạo trông như đến từ tổ chức uy tín, **lừa** người dùng bấm vào liên kết giả hoặc tự nhập thông tin cá nhân/tài chính vào website giả — dựa hoàn toàn vào việc **đánh lừa người dùng**. **Pharming**: dùng mã độc hoặc phá hoại bản ghi DNS/file hosts để **tự động chuyển hướng** người dùng sang website giả, **ngay cả khi họ gõ đúng địa chỉ thật** — không cần đánh lừa người dùng tại thời điểm tấn công, việc chuyển hướng diễn ra âm thầm và tự động.

(!) Lỗi thường gặp: The single clearest distinguishing test in an exam answer: **did the user have to be tricked into clicking something or entering something (phishing)**, or **did the redirection happen automatically even though the user did everything correctly, including typing the right URL (pharming)**? If the correct URL was typed and the user still ended up somewhere fraudulent, this is pharming, not phishing.

Ví dụ mẫu · Worked example

A user receives an email claiming to be from their bank, warning that their account will be suspended unless they click a link and "verify" their password within 24 hours. Identify the attack, and explain the reasoning a security-aware user should apply.

This is **phishing**. The email is designed to look legitimate and creates a false sense of urgency, aiming to **trick** the user into clicking the link and voluntarily typing their password into what is actually a fake site controlled by the attacker. A security-aware user should be suspicious of urgent, unsolicited requests for credentials, should check the sender's actual email address and the link's actual destination URL rather than trusting the displayed text, and should instead navigate to the bank's website directly (e.g. by typing the known address themselves) rather than clicking the link in the email.

5.3.3 Brute-force attacks

A **brute-force attack** uses software to **systematically try every possible combination** of characters (or, in a related variant, every word in a large dictionary of common passwords) as a login password, one attempt after another, until the correct one is eventually found. It does not rely on tricking the user at all — it relies purely on trying enough combinations, given enough time and processing power.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tấn công dò mật khẩu (brute-force): dùng phần mềm **thử lần lượt mọi tổ hợp ký tự** có thể (hoặc mọi từ trong một danh sách mật khẩu phổ biến) cho đến khi tìm ra đúng mật khẩu — không cần lừa người dùng, chỉ dựa vào việc thử đủ nhiều tổ hợp với đủ thời gian và sức mạnh tính toán.

Two measures directly counter a brute-force attack: (1) requiring **long, complex passwords** (mixing upper- and lower-case letters, digits and symbols) massively increases the *number* of possible combinations that must be tried, making an exhaustive search take impractically long; and (2) **locking an account** (temporarily or permanently) after a small number of consecutive failed login attempts prevents an attacker's software from simply trying attempt after attempt without limit.

VN **GIẢI THÍCH TIẾNG VIỆT**

Hai biện pháp chống brute-force: (1) yêu cầu **mật khẩu dài, phức tạp** (chữ hoa, chữ thường, số, ký tự đặc biệt) làm tăng vọt số tổ hợp cần thử, khiến việc dò tìm mất thời gian bất khả thi; (2) **khoá tài khoản** sau một số lần đăng nhập sai liên tiếp, ngăn phần mềm tấn công thử liên tục không giới hạn.

Ví dụ mẫu · Worked example

A website allows unlimited login attempts with no delay or lockout, and accepts short, simple passwords (4 lower-case letters only). Explain why this website is especially vulnerable to a brute-force attack, and recommend two specific changes.

With only lower-case letters and a length of 4, there are only $26^4 = 456{,}976$ possible passwords — a trivially small number for automated software to work through, especially with no limit on how many attempts can be made per second. **Recommendation 1:** require longer, more complex passwords (mixing upper/lower case, digits, symbols, and a greater minimum length), which increases the number of possible combinations enormously, making exhaustive search take far too long to be practical. **Recommendation 2:** lock the account (e.g. for a period of time, or until reset by the user) after a small fixed number of consecutive failed attempts, so automated software cannot keep guessing indefinitely.

5.3.4 Data interception and packet sniffing

Data interception (packet sniffing) occurs when an attacker uses special software (a "packet sniffer") to **capture data packets** as they travel across a network, in order to read whatever information those packets contain. This is a particular risk on **unsecured public Wi-Fi** (e.g. in a café or airport), where any nearby device running suitable software can potentially capture packets travelling over the same wireless network. If the intercepted data is sent **unencrypted**, the attacker can read it directly (e.g. login credentials, personal messages); encryption is therefore the key defence, since it means the attacker only captures unreadable ciphertext.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chặn dữ liệu (data interception / packet sniffing): kẻ tấn công dùng phần mềm "bắt gói tin" để **chặn thu các gói dữ liệu** khi chúng di chuyển qua mạng — rủi ro cao khi dùng **Wi-Fi công cộng không bảo mật**. Nếu dữ liệu bị chặn **không được mã hoá**, kẻ tấn công có thể đọc trực tiếp (mật khẩu, tin nhắn cá nhân); vì vậy **mã hoá** là biện pháp phòng vệ quan trọng nhất, vì khi đó kẻ tấn công chỉ thu được dữ liệu vô nghĩa.

Ví dụ mẫu · Worked example

A traveller logs into their email account while connected to a free, unsecured public Wi-Fi network at an airport. Explain the risk this creates, and state one effective way to reduce it.

Risk: an attacker connected to the same unsecured Wi-Fi network could run packet-sniffing software to capture data packets travelling across that network. If the traveller's login data is sent without encryption, the attacker could directly read the captured packets and obtain the traveller's username and password.

Reducing the risk: using a connection that encrypts the data (e.g. ensuring the email service is accessed only over **HTTPS**, or using a **VPN** to encrypt all traffic leaving the device) means that, even if an attacker successfully intercepts the packets, the captured data is encrypted and unreadable without the correct decryption key.

5.3.5 SQL injection

Many websites store their data (user accounts, product catalogues, orders) in a database, and build a query to that database using text the user themselves types into a web form (for example, a username typed into a login box). If the website does not carefully check that input first, an attacker can exploit this.

SQL injection is an attack in which an attacker deliberately enters **malicious SQL code** into a web form's input field (rather than the ordinary data the field expects), intending that code to be executed by the underlying database as part of its query — potentially bypassing authentication, or extracting, altering or deleting data the attacker should never have been able to access.

VN GIẢI THÍCH TIẾNG VIỆT

SQL injection: kẻ tấn công cố tình nhập **mã SQL độc hại** vào ô nhập liệu của một biểu mẫu web (thay vì dữ liệu bình thường), với mục đích khiến cơ sở dữ liệu bên dưới **thực thi đoạn mã đó** như một phần của câu truy vấn — có thể vượt qua khâu xác thực đăng nhập, hoặc trích xuất/sửa đổi/xoá dữ liệu mà kẻ tấn công lẽ ra không được phép truy cập.

Ví dụ mẫu · Worked example

A login form builds the following database query directly from what the user types into the **username** and **password** boxes:

```
SELECT * FROM users WHERE username='<username>' AND password='<password>';
```

A legitimate user named `admin` would type `admin` into the username box, causing the harmless query:

```
SELECT * FROM users WHERE username='admin' AND password='letmein123';
```

Instead, an attacker who does not know the password types `admin'--` into the **username** box (and anything at all into the password box). Explain what the resulting query becomes, and why this lets the attacker log in without knowing the password.

Substituting this input directly into the query text produces:

```
SELECT * FROM users WHERE username='admin'--' AND password='anything';
```

In SQL, the two hyphens `--` mark the **start of a comment**, so the database engine ignores everything on the line from that point onward — including the entire `AND password='anything'` condition. The query the database actually executes is therefore effectively:

```
SELECT * FROM users WHERE username='admin'
```

which matches the `admin` account **regardless of any password at all**, letting the attacker log in as `admin` without ever knowing the real password.

SQL injection is prevented primarily by **input validation and sanitisation**: the website checks (and, where necessary, rejects or "escapes") anything entered into a form field that is not consistent with the kind of data expected (e.g. rejecting quote marks, semicolons or comment markers in a field that should only ever contain a plain username), so user input can never be interpreted as executable SQL code. Well-designed systems additionally use **parameterised queries** (also called prepared statements), which treat everything a user types strictly as *data* to be matched, never as part of the executable query structure itself — so even deliberately malicious-looking input has no special effect on the query at all.

VN GIẢI THÍCH TIẾNG VIỆT

Cách ngăn chặn SQL injection chủ yếu là **kiểm tra và làm sạch dữ liệu đầu vào (input validation/sanitisation)**: website kiểm tra (và từ chối hoặc "escape") bất kỳ nội dung nhập vào không đúng định dạng mong đợi (ví dụ từ chối dấu nháy, dấu chấm phẩy, ký hiệu comment trong ô chỉ nên chứa tên đăng nhập thuần). Hệ thống thiết kế tốt còn dùng **câu truy vấn tham số hoá (parameterised queries)**, luôn coi dữ liệu người dùng nhập là **dữ liệu thuần**, không bao giờ là một phần cấu trúc câu lệnh thực thi.

(!) Lỗi thường gặp: Do not describe SQL injection as simply "entering a wrong password". The attack specifically depends on the input containing characters with **special meaning in SQL** (quote marks, comment markers, logical operators) that change the *structure* of the query itself — an ordinary wrong password, with no special characters, would simply fail to match, exactly as intended.

5.3.6 Denial-of-service (DoS) attacks

A **denial-of-service (DoS) attack** floods a target server with an overwhelming volume of traffic or requests, deliberately consuming its available bandwidth, processing capacity or memory, so that the server becomes too overloaded to respond to **legitimate** users' requests in a reasonable time (or at all). When the flood of traffic is generated from **many different computers at once** (often a large network of previously-infected, attacker-controlled machines), the attack is called a **distributed denial-of-service (DDoS)** attack.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tấn công từ chối dịch vụ (DoS): làm ngập máy chủ mục tiêu bằng một lượng lớn lưu lượng truy cập/yêu cầu, cố tình chiếm hết băng thông, khả năng xử lý hoặc bộ nhớ của máy chủ, khiến máy chủ quá tải và **không thể phục vụ** người dùng hợp lệ. Khi lưu lượng tấn công đến từ **nhiều máy tính khác nhau cùng lúc**, đây gọi là tấn công **từ chối dịch vụ phân tán (DDoS)**.

Ví dụ mẫu · Worked example

An online ticket-booking website suddenly becomes unreachable for genuine customers moments after tickets for a popular event go on sale, even though the company's servers have not failed or been physically damaged. Investigators find an enormous, sudden spike in traffic arriving from thousands of different IP addresses at once, none of which ever completes a real purchase. Explain what has most likely happened.

This is most likely a **distributed denial-of-service (DDoS) attack**. Thousands of separate, attacker-controlled devices have simultaneously sent an overwhelming volume of requests to the ticket-booking server, deliberately consuming its bandwidth and processing capacity. Because the server's resources are exhausted responding to (or simply receiving) this flood of illegitimate traffic, it has no capacity left to respond to genuine customers trying to buy tickets — the service has effectively been "denied" to legitimate users, even though nothing about the server's own hardware or software has actually broken.

5.3.7 Threats at a glance

Threat	How it works	One defence
Virus	Attaches to a file; needs the file opened/run by a user to activate and spread.	Anti-malware software; avoid opening unknown attachments/files.
Worm	Self-replicates and spreads across a network automatically, with no host file and no user action needed.	Firewall to block unauthorised network access; keep software patched.
Trojan horse	Disguised as legitimate software; user is tricked into installing it themselves; opens a hidden back door.	Only install software from trusted sources; anti-malware scanning.
Spyware	Secretly monitors and transmits user activity (e.g. keystrokes) to a third party.	Anti-malware software; avoid untrusted downloads.
Ransomware	Encrypts the victim's files and demands payment for restored access.	Regular offline/disconnected backups, so files can be restored without paying.
Phishing	Fraudulent message tricks the user into clicking a link / entering personal data.	User awareness/training; check sender and link destination before clicking.
Pharming	Malicious code/DNS corruption automatically redirects the user, even to a correctly-typed URL.	Anti-malware software; checking for a valid digital certificate/padlock.
Brute-force	Systematically tries every possible password combination until correct.	Long, complex passwords; account lockout after repeated failed attempts.
Data interception	Attacker captures unencrypted data packets travelling across a network.	Encryption (HTTPS, VPN); avoid unsecured public Wi-Fi for sensitive tasks.
SQL injection	Malicious SQL code entered into a web form manipulates/extracts database data.	Input validation/sanitisation; parameterised queries.
Denial-of-service (DoS/DDoS)	Server flooded with excessive traffic so it cannot serve legitimate users.	Firewalls/traffic filtering; rate limiting.

Cyber security threats: mechanism and one representative defence for each.

5.4 Protection measures

No single measure defends against every threat above; in practice, several protection measures are used together, each targeting a different weak point.

5.4.1 Encryption, firewalls, anti-malware, passwords and 2FA

Khái niệm cốt lõi

SSL/TLS encryption (HTTPS): encrypts data travelling between a browser and a web server, so that even if it is intercepted (e.g. by packet sniffing), it cannot be read without the correct decryption key. A secure connection is indicated in the browser by the **padlock icon** and a URL beginning `https://` rather than plain `http://`.

Firewall: software (or a dedicated hardware device) that monitors and filters incoming and outgoing network traffic against a set of rules, blocking traffic that appears unauthorised or suspicious while allowing legitimate traffic through — a key defence against worms and unauthorised remote access, and one useful tool against denial-of-service traffic.

Anti-malware/antivirus software: scans files and running programs against a database of known malware signatures (plus heuristic detection of suspicious, previously-unseen behaviour), quarantining or deleting anything infected.

Strong, unique passwords: long passwords mixing upper/lower-case letters, digits and symbols dramatically increase the number of combinations a brute-force attack must try; using a *different* password per site limits the damage if one password is ever compromised.

Two-factor authentication (2FA): requires a user to provide **two** different types of evidence of identity before being logged in — typically something they **know** (a password) combined with something they **have** (a one-time code sent to their phone) or something they **are** (a fingerprint). Even if an attacker discovers the password, they still cannot log in without the second factor.

VN GIẢI THÍCH TIẾNG VIỆT

SSL/TLS (HTTPS): mã hoá dữ liệu giữa trình duyệt và máy chủ, hiển thị bằng **biểu tượng ổ khoá** và `https://`. **Firewall**: giám sát và lọc lưu lượng mạng ra/vào theo bộ quy tắc, chặn truy cập trái phép. **Anti-malware**: quét file/chương trình theo cơ sở dữ liệu chữ ký mã độc đã biết. **Mật khẩu mạnh**: dài, đa dạng ký tự, khác nhau cho từng tài khoản. **Xác thực hai yếu tố (2FA)**: yêu cầu hai bằng chứng danh tính khác loại (ví dụ mật khẩu + mã gửi về điện thoại), nên kẻ tấn công biết mật khẩu vẫn không đăng nhập được nếu thiếu yếu tố thứ hai.

Ví dụ mẫu · Worked example

A customer is about to type their credit card number into an online checkout form. Explain what they should check in the browser before doing so, and what protection this provides.

The customer should check that the address bar shows a **padlock icon** and that the URL begins with `https://` (not `http://`). This confirms the connection to the site uses **SSL/TLS encryption**: the credit card number will be encrypted before it leaves the customer's browser, so that even if the data is intercepted while travelling across the network (e.g. by packet sniffing on a shared network), an attacker would only capture unreadable, encrypted ciphertext rather than the actual card number.

5.4.2 Proxy servers

A **proxy server** sits as an **intermediary** between a user's device and the wider Internet: instead of the user's device contacting a web server directly, its requests are first sent to the proxy server, which forwards them on (and forwards the response back). Because every request passes through it, a proxy server can: **filter requests** (e.g. blocking access to specified categories of website, useful in a school or workplace); **hide the user's real IP address** from the websites they visit, since those websites only ever see the proxy server's IP address, not the user's own; and **cache** frequently-requested pages (storing a local copy), so that later requests for the same page can be served faster, without needing to fetch it again from the original web server.

VN GIẢI THÍCH TIẾNG VIỆT

Proxy server đóng vai trò **trung gian** giữa thiết bị người dùng và Internet: mọi yêu cầu đi qua proxy trước khi đến đích. Nhờ vậy proxy có thể: **lọc yêu cầu** (chặn một số loại website, ví dụ tại trường học/công ty); **ẩn địa chỉ IP thật** của người dùng (website đích chỉ thấy IP của proxy); và **lưu đệm (cache)** các trang được yêu cầu thường xuyên để phục vụ nhanh hơn cho lần sau.

Ví dụ mẫu · Worked example

A school connects all student devices to the Internet through a proxy server. Explain two distinct benefits this gives the school.

Filtering: the proxy server can be configured to block requests to categories of website the school does not wish students to access (e.g. social media or inappropriate content), since every student request passes through it and can be checked against a set of rules before being allowed through.

Caching: when many students request the same commonly-used educational page in a short period, the proxy server can store (cache) a local copy after the first request and serve that copy directly to later requests, reducing the school's external bandwidth usage and speeding up access for students, instead of fetching the same page repeatedly from the original web server.

5.4.3 Biometric authentication

Biometric authentication verifies a user's identity using a measurable, unique **physical characteristic** — most commonly a **fingerprint**, **facial recognition**, or an **iris/retina scan**. Because these characteristics are inherent to the individual's own body, they are considerably harder for an attacker to guess, steal by observation, or simply forget than a traditional typed password, though a captured/copied biometric sample can, in principle, still be misused, and (unlike a password) a compromised biometric characteristic cannot simply be "changed".

VN **GIẢI THÍCH TIẾNG VIỆT**

Xác thực sinh trắc học xác minh danh tính người dùng bằng một **đặc điểm sinh học riêng biệt** — phổ biến nhất là **vân tay**, **nhận diện khuôn mặt**, hoặc **quét mống mắt/võng mạc**. Vì các đặc điểm này gắn liền với cơ thể, chúng khó bị đoán, quan sát trộm hay quên hơn nhiều so với mật khẩu — tuy nhiên nếu dữ liệu sinh trắc học bị đánh cắp/sao chép thì (không như mật khẩu) người dùng không thể đơn giản "đổi" đặc điểm sinh học của mình.

Ví dụ mẫu · Worked example

A company replaces its staff login system, which previously required only a typed password, with a system requiring a fingerprint scan. Explain one advantage and one limitation of this change.

Advantage: a fingerprint cannot be forgotten (unlike a password) and is far harder for an attacker to guess or simply observe being entered (e.g. by watching someone type), since it is a physical characteristic unique to each employee rather than a memorised string of characters.

Limitation: if an employee's fingerprint data is ever compromised (e.g. stolen from a poorly-secured database, or convincingly replicated), the employee cannot simply "change" their fingerprint the way they could change a leaked password — the compromised biometric characteristic remains a permanent vulnerability for that individual.

5.4.4 Digital certificates

A **digital certificate** is issued by a trusted, independent **Certificate Authority (CA)** and verifies that a particular website's stated **public key** genuinely belongs to that website (and not to an impostor pretending to be it). When a browser connects to a website over HTTPS, the website presents its digital certificate; the browser checks that the certificate is valid, has not expired, matches the domain being visited, and was signed by a CA the browser already trusts. Only once this check succeeds does the browser establish the encrypted SSL/TLS connection and display the padlock icon — digital certificates are therefore the mechanism that underpins *why* a user can trust an HTTPS connection in the first place.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chứng chỉ số (digital certificate) do một **tổ chức cấp chứng chỉ (CA)** đáng tin cậy cấp phát, xác nhận **khoá công khai (public key)** của một website thực sự thuộc về website đó (không phải kẻ mạo danh). Khi trình duyệt kết nối HTTPS, website gửi chứng chỉ số; trình duyệt kiểm tra chứng chỉ còn hiệu lực, đúng tên miền, và được ký bởi một CA đáng tin cậy — chỉ khi đó trình duyệt mới thiết lập kết nối mã hoá SSL/TLS và hiển thị biểu tượng ổ khoá. Đây chính là cơ chế làm nền tảng cho lòng tin vào kết nối HTTPS.

Ví dụ mẫu · Worked example

Explain why a browser displaying a padlock icon for a website depends on more than encryption alone.

Encryption by itself only guarantees that data cannot be *read* by an eavesdropper while in transit — it says nothing about *who* is actually on the other end of that encrypted connection. A **digital certificate**, issued by a trusted **Certificate Authority**, additionally verifies that the public key being used to set up the encrypted connection genuinely belongs to the website the user intended to visit. The browser checks this certificate (validity, expiry, matching domain, trusted issuer) *before* showing the padlock, so the padlock actually certifies two things at once: the connection is encrypted, *and* the site's identity has been verified by an independent trusted authority — without the certificate check, a user could unknowingly set up a perfectly encrypted connection directly with an impostor.

5.4.5 Protection measures at a glance

Measure	What it protects against / how
SSL/TLS (HTTPS)	Encrypts data in transit between browser and server; defeats data interception; shown via padlock icon.
Firewall	Filters incoming/outgoing network traffic by rule; blocks unauthorised access and some malicious traffic.
Anti-malware software	Scans and removes viruses, worms, trojans, spyware and ransomware using signatures/heuristics.
Strong, unique passwords	Increases combinations a brute-force attack must try; limits damage if one account is compromised.
Two-factor authentication (2FA)	Requires a second, independent factor beyond the password, defeating a stolen/guessed password alone.
Proxy server	Filters requests, hides the user's real IP address, and caches frequently-requested pages.
Biometric authentication	Uses a unique physical characteristic (fingerprint, face, iris) as a login factor, hard to guess or forget.
Digital certificate	Issued by a trusted CA; verifies a website's public key truly belongs to that website, underpinning SSL/TLS trust.
Input validation/sanitisation	Rejects or escapes unexpected characters in form fields, defeating SQL injection.
Offline/regular backups	Restores data after ransomware or data loss without needing to pay any ransom.

Protection measures and the threat(s) each is chiefly aimed at.

5.5 Practice

Bài tập luyện tập

In the URL `https://www.school.edu/exams/timetable.pdf`, what does `https` indicate?

(A) The domain name of the website

(B) The protocol used to transfer the page, here with encryption

(C) The folder in which the file is stored

(D) The name of the specific file being requested

Lời giải chi tiết

The first part of a URL, ending in `://`, states the **protocol**. `https` specifically indicates HTTP transferred over an encrypted (secure) connection. **(B)**.

Bài tập luyện tập

A user receives an email, apparently from their bank, asking them to click a link and confirm their account password within 24 hours or face suspension. Which type of attack does this describe?

(A) Pharming

(B) Phishing

(C) A denial-of-service attack

(D) A brute-force attack

Lời giải chi tiết

The message is designed to **trick** the user into clicking a link and entering their password — this reliance on deceiving the user is the defining feature of **phishing**. **(B)**.

Bài tập luyện tập

A user types their bank's correct web address directly into the address bar, exactly as always, but is silently redirected to a fraudulent site that looks identical to the real one. Investigation reveals the DNS cache on the user's router had been corrupted. Which type of attack does this describe?

(A) Phishing
(B) Pharming
(C) A trojan horse
(D) Spyware

Lời giải chi tiết

The user typed the **correct URL** and was still redirected, automatically and without being tricked into clicking anything — this is the defining feature of **pharming**, here caused by corruption of DNS information. **(B)**.

Bài tập luyện tập

Explain, with reference to encryption and the browser's address bar, why a customer should check for a padlock icon before entering a credit card number on a checkout page.

Lời giải chi tiết

The padlock icon indicates the connection is using **SSL/TLS encryption** (shown by the URL beginning `https://`). This means the credit card number is encrypted by the browser before it leaves the customer's device, so that if the data is intercepted while travelling across the network, an attacker only captures encrypted, unreadable ciphertext rather than the actual card number. Without the padlock (a plain `http://` connection), the data would travel unencrypted, and anyone able to intercept it (e.g. via packet sniffing on a shared network) could read the card number directly.

Bài tập luyện tập

A company's files are suddenly all encrypted, and a message demands payment in cryptocurrency for the decryption key. (a) Identify the type of malware. (b) Explain one measure that would let the company recover its data without paying.

Lời giải chi tiết

(a) This is **ransomware** — malware that encrypts a victim's files and demands payment for the means to restore access.
(b) If the company maintains regular, **offline (disconnected) backups** of its data, it can simply restore its files from a backup made before the attack, without needing the attacker's decryption key or paying any ransom. (The backup must be offline/disconnected, since a backup left permanently connected to the network could potentially be encrypted by the same ransomware attack.)

Bài tập luyện tập

Explain the difference between the **Internet** and the **World Wide Web**, using email as part of your answer.

Lời giải chi tiết

The **Internet** is the physical, global network of interconnected devices and communication links that can carry many different kinds of data for many different services. The **World Wide Web** is just one of those services: an interlinked collection of webpages, accessed via HTTP/HTTPS through a browser. Email demonstrates the distinction clearly: sending an email uses the **Internet** (the underlying network) to travel from sender to recipient, but it is *not* part of the Web, because it uses entirely different protocols (such as SMTP/IMAP) rather than HTTP/HTTPS, and does not involve webpages or hyperlinks.

Bài tập luyện tập

What is the role of a DNS server?

(A) It stores a website's actual files and sends them to browsers

(B) It translates a domain name into the numerical IP address of the server hosting it

(C) It encrypts data sent between a browser and a web server

(D) It filters a user's requests and hides their IP address

Lời giải chi tiết

A DNS server holds (or looks up) a database mapping domain names to IP addresses, so a browser can find the correct server to contact. **(B)**. (Option A describes a web server/hosting; option C describes SSL/TLS; option D describes a proxy server.)

Bài tập luyện tập

Explain what is meant by **hosting** a website, and what an IP address is used for in this process.

Lời giải chi tiết

Hosting is the service of storing a website's files (HTML, images, scripts, etc.) on a web server that is permanently connected to the Internet, so the site is available to visitors at any time. Every web server needs an **IP address** — a numerical address that uniquely identifies it on the network — so that requests sent from a browser (once resolved from the site's domain name via DNS) can actually be delivered to the correct physical server among the millions connected to the Internet.

Bài tập luyện tập

List, in the correct order, the four main things that happen when a user types a new domain name into their browser for the first time: (i) the web server sends the page back, (ii) the browser sends an HTTP/HTTPS request to the resolved IP address, (iii) the browser sends the domain name to a DNS server, (iv) the DNS server returns the corresponding IP address.

Lời giải chi tiết

Correct order: **(iii)** the browser sends the domain name to a DNS server → **(iv)** the DNS server returns the corresponding IP address → **(ii)** the browser sends an HTTP/HTTPS request to that resolved IP address → **(i)** the web server sends the requested page back to the browser.

Bài tập luyện tập

An online store uses a cookie to remember the items in a customer's shopping cart, but only while the customer keeps their browser open during a single visit — the cart is empty again the next time they visit, even minutes after closing the browser. Identify the type of cookie, and justify your answer.

Lời giải chi tiết

This is a **session cookie**. A session cookie is held only in the browser's memory for the duration of one browsing session and is automatically deleted the moment the browser closes. Because the cart contents do not survive the browser being closed — even for a very short time — this rules out a persistent cookie, which would instead be saved to disk with an expiry date and would still be present the next time the store was visited.

Bài tập luyện tập

A user logs into a website once, ticks "remember me", closes their browser, and finds themselves still logged in automatically three weeks later. Which type of cookie explains this behaviour, and why?

(A) Session cookie, because it is stored in RAM only

(B) Persistent cookie, because it is saved to disk with an expiry date and survives the browser being closed

(C) Neither — this requires a digital certificate

(D) Session cookie, because "remember me" always uses temporary storage

Lời giải chi tiết

Login information surviving the browser being closed, for weeks at a time, can only be explained by a **persistent cookie**: it is written to disk (not just held in memory) with a stored expiry date, so it remains available across many separate browsing sessions until it expires or is deleted. **(B)**.

Bài tập luyện tập

Give one example of data a **session** cookie is typically used to store, and one example of data a **persistent** cookie is typically used to store. Explain why each choice suits that cookie type.

Lời giải chi tiết

Session cookie example: the contents of a shopping cart while a customer continues browsing an online store during one visit. This suits a session cookie because the cart only needs to exist while the customer is actively shopping right now; it is reasonable for it to disappear once the browser (and that visit) ends.

Persistent cookie example: a saved login, so a returning user is automatically signed in without retyping their password on a later visit. This suits a persistent cookie because the whole point is for the site to recognise the same user again *after* the browser has been closed and reopened, potentially days or weeks later — which requires the cookie to be written to disk with an expiry date, rather than disappearing at the end of the current session.

Bài tập luyện tập

Which of the following best distinguishes a computer **virus** from a computer **worm**?

(A) A virus deletes data; a worm never deletes data

(B) A virus requires a host file to be opened by a user to spread; a worm spreads across a network automatically with no user action needed

(C) A worm only affects one computer; a virus spreads across many computers

(D) There is no meaningful difference between the two

Lời giải chi tiết

A virus attaches to a file and needs that file opened/run by a user before it can activate and spread; a worm self-replicates and spreads across a network entirely on its own, without a host file and without any user action. **(B)**.

Bài tập luyện tập

An employee installs what they believe is a free productivity tool downloaded from an unofficial website. The software works as advertised, but an attacker later gains remote control of the employee's computer through a channel the tool secretly opened during installation. Identify the malware type and justify your answer.

Lời giải chi tiết

This is a **trojan horse**. The malicious program disguised itself as a genuine, desirable piece of software (a productivity tool) so that the employee would install it **voluntarily** — it did not need to self-replicate or exploit a network vulnerability to spread. Once installed, it performed a hidden, harmful action (opening a back door for remote access) that had nothing to do with its advertised purpose, which is the defining behaviour of a trojan.

Bài tập luyện tập

A parent notices their child's laptop has become slow, and a security scan reveals a program has been silently recording every key pressed on the keyboard, including passwords, and sending this information to an unknown remote server. Identify the malware type.

(A) Worm

(B) Ransomware

(C) Spyware

(D) Denial-of-service software

Lời giải chi tiết

Secretly monitoring user activity (here, keystrokes) and transmitting the gathered information to a third party without the user's knowledge is the defining behaviour of **spyware**. **(C)**.

Bài tập luyện tập

A login page has no limit on the number of attempts a user (or a piece of automated software) may make, and accepts passwords as short as three digits. Explain why this makes the system vulnerable to a brute-force attack, and describe two specific changes that would reduce the risk.

Lời giải chi tiết

With only digits and a length of three, there are just $10^3 = 1000$ possible passwords — a number automated software could work through in a very short time, especially with no restriction on how many attempts can be made. This makes an exhaustive, systematic search (a **brute-force attack**) highly likely to succeed quickly.

Change 1: require longer, more complex passwords (mixing upper/lower-case letters, digits and symbols, with a greater minimum length), which increases the total number of possible combinations enormously, making an exhaustive search take impractically long.
Change 2: lock the account (temporarily, or until a reset process is completed) after a small number of consecutive failed login attempts, preventing automated software from making unlimited guesses.

Bài tập luyện tập

Which of the following best describes data interception (packet sniffing)?

(A) Flooding a server with traffic so it cannot respond to legitimate users

(B) Capturing data packets as they travel across a network in order to read their contents

(C) Entering malicious code into a web form's input field

(D) Systematically trying every possible password combination

Lời giải chi tiết

Data interception uses packet-sniffing software to capture data packets in transit across a network, so their contents can be read (especially if sent unencrypted). **(B).** (The other options describe a DoS attack, SQL injection, and a brute-force attack respectively.)

Bài tập luyện tập

A remote worker regularly accesses their company's internal systems while connected to free, unsecured Wi-Fi networks in cafes. Explain the specific risk this creates and recommend one effective countermeasure.

Lời giải chi tiết

Risk: an attacker connected to the same unsecured Wi-Fi network could use packet-sniffing software to capture data packets travelling across that network. If the worker's data (e.g. login credentials or company documents) is transmitted without encryption, the attacker could read the captured packets directly and obtain sensitive information.

Countermeasure: the worker should ensure all traffic is encrypted, for example by only accessing company systems over **HTTPS** and/or using a **VPN**, which encrypts all data leaving the device — so that even if packets are successfully intercepted, they are unreadable without the correct decryption key.

Bài tập luyện tập

A login form builds its database query directly from unvalidated user input. An attacker types `admin'--` into the username field. Explain why this specific input can allow the attacker to log in without knowing the password, and state one general technique that would prevent this.

Lời giải chi tiết

The two hyphens `--` form a **comment marker** in SQL, so once this text is inserted directly into the query, everything appearing after it on that line — including the `AND password=...` check — is treated as a comment and **ignored** by the database engine. The query effectively executed becomes a simple match on `username='admin'` alone, with no password check at all, letting the attacker log in as `admin` regardless of what (if anything) was typed as the password.

Prevention: the website should apply **input validation/sanitisation** (rejecting or escaping characters such as quote marks and comment markers that have no legitimate place in a plain username) and, more robustly, use **parameterised queries/prepared statements**, which always treat user-supplied text strictly as data, never as part of the executable query structure — so this input would simply fail to match any real username.

Bài tập luyện tập

Which of the following is the best explanation of why parameterised queries (prepared statements) prevent SQL injection?

(A) They make the database run faster than an ordinary query

(B) They encrypt all data stored in the database

(C) They ensure user-supplied input is always treated strictly as data, never as part of the executable query structure

(D) They automatically create daily backups of the database

Lời giải chi tiết

Parameterised queries separate the fixed query structure from the values supplied by the user, so that whatever a user types — however it is formatted, and even if it contains characters such as quote marks or comment markers — is always substituted purely as a **data value**, never interpreted as part of the SQL command itself. **(C)**.

Bài tập luyện tập

A large retailer's website becomes completely unreachable for two hours after receiving an abnormally enormous volume of simultaneous requests from thousands of different sources, none of which represent genuine customers. Identify the type of attack, and explain how it achieves its effect.

Lời giải chi tiết

This describes a **denial-of-service attack** (specifically, given the many different simultaneous sources, a **distributed** denial-of-service, or DDoS, attack). It achieves its effect by flooding the server with such an overwhelming volume of traffic/requests that the server's bandwidth, processing capacity or memory becomes fully consumed responding to (or merely receiving) the flood — leaving no capacity for the server to respond to genuine customers' requests within a reasonable time, effectively denying them the service even though the server itself has not been physically damaged.

Bài tập luyện tập

Which combination of measures would best protect against a brute-force attack specifically?

(A) SSL/TLS encryption and a digital certificate

(B) Long, complex passwords and locking an account after repeated failed login attempts

(C) A proxy server and biometric authentication

(D) Disk defragmentation and file compression

Lời giải chi tiết

Brute-force attacks are defeated by making the number of combinations to try impractically large (long, complex passwords) and by preventing unlimited automated guessing (locking the account after a small number of consecutive failed attempts). **(B)**.

Bài tập luyện tập

Explain how a firewall helps protect a home network, giving one example of traffic it might legitimately allow and one example of traffic it might block.

Lời giải chi tiết

A **firewall** monitors all incoming and outgoing network traffic and compares it against a set of configured rules, allowing traffic that matches permitted patterns through and blocking traffic that does not. **Example allowed:** a request the user's own browser sends out to load a webpage, and the corresponding reply coming back, matches an expected, legitimate pattern and is allowed through. **Example blocked:** an unsolicited connection attempt arriving from an unknown external device, trying to access the network without ever having been requested by anything inside it, does not match a legitimate pattern and is blocked — this is exactly the kind of unauthorised access a worm might attempt to exploit.

Bài tập luyện tập

A company wants every employee login to require both a password and a one-time code sent to the employee's personal phone. What is this security measure called, and why is it more secure than a password alone?

(A) Biometric authentication

(B) Two-factor authentication (2FA)

(C) A digital certificate

(D) A proxy server

Lời giải chi tiết

Requiring two different types of evidence of identity — something the employee **knows** (the password) and something they **have** (their phone, receiving the one-time code) — is **two-factor authentication (2FA)**. It is more secure than a password alone because even if an attacker discovers or guesses the password (e.g. through a brute-force or phishing attack), they still cannot log in without also possessing the employee's phone to receive the one-time code. **(B)**.

Bài tập luyện tập

A school wants students to be able to log in using a fingerprint scanner instead of typing a password. Give one advantage and one limitation of this approach compared to password-based login.

Lời giải chi tiết

Advantage: a fingerprint cannot be forgotten, and is much harder for another student to guess, observe being entered, or share (deliberately or accidentally) than a typed password, since it is a physical characteristic unique to each individual.

Limitation: if fingerprint data is ever stolen or convincingly replicated, the affected student cannot simply "change" their fingerprint the way a compromised password can be changed — unlike a leaked password, a compromised biometric characteristic represents a permanent vulnerability for that individual.

Bài tập luyện tập

Explain the role of a Certificate Authority (CA) in establishing trust in an HTTPS connection, and what the browser checks before displaying the padlock icon.

Lời giải chi tiết

A **Certificate Authority (CA)** is a trusted, independent organisation that issues **digital certificates**, each verifying that a specific website's stated public key genuinely belongs to that website. When a browser connects to a site over HTTPS, the site presents its digital certificate; the browser checks that the certificate has not expired, that it matches the domain actually being visited, and that it was signed by a CA the browser already trusts. Only if all of these checks succeed does the browser establish the encrypted SSL/TLS connection and display the **padlock icon** — without this check, a user could end up with a perfectly encrypted connection directly to an impostor site, since encryption alone does not verify *who* is on the other end.

Bài tập luyện tập

A company connects all of its office computers to the Internet through a proxy server. Identify the three distinct functions a proxy server can provide, illustrating each with a brief example relevant to this office.

Lời giải chi tiết

Filtering: the proxy server can block requests to categories of website the company does not permit during work hours (e.g. streaming video sites), since every employee's request passes through it and can be checked against configured rules.

Hiding the real IP address: external websites the employees visit see only the proxy server's IP address, not each individual employee's own device IP address, adding a layer of anonymity for the office network.

Caching: if many employees request the same frequently-used external resource (e.g. a common software update or reference page) within a short period, the proxy server can store a local copy after the first request and serve it directly to later requests, reducing external bandwidth use and speeding up access.

Bài tập luyện tập

A small business owner is auditing their online security and lists four incidents from the past year: (i) an employee's laptop was infected after opening an email attachment, and the infection copied itself unaided to two other computers on the office network overnight; (ii) a fake invoice email tricked an employee into entering company bank login details on a convincing but fraudulent website; (iii) the business's customer database was queried in an unexpected way after a customer typed unusual text into the "search product" box; (iv) the business's public website became briefly unreachable during an unusually large, sudden spike in traffic from thousands of unfamiliar sources. For each incident, name the most likely type of attack.

Lời giải chi tiết

(i) This behaviour combines a **virus** (it needed the email attachment to be opened before it activated) with **worm**-like self-replication once active (it then copied itself unaided across the network) — in Cambridge IGCSE terms, the initial infection method (an opened attachment) is characteristic of a virus, while its subsequent automatic, unaided spread across the network is characteristic of worm behaviour.

(ii) This is **phishing** — a fraudulent message impersonating a trusted source tricked the employee into voluntarily entering login details into a fake website.

(iii) This is most likely **SQL injection** — unusual text entered into a web form's input field appears to have been interpreted as part of the underlying database query, rather than as ordinary search text.

(iv) This is a **denial-of-service (DDoS) attack** — an overwhelming, sudden volume of traffic from many different sources consumed the website's resources, making it briefly unreachable for genuine visitors.

Automated and Emerging Technologies

Mục tiêu Unit: Rô-bốt và hệ thống điều khiển: vòng lặp phản hồi cảm biến → bộ xử lý → cơ cấu chấp hành; so sánh rô-bốt công nghiệp với người lao động; trí tuệ nhân tạo: học máy (machine learning) và hệ chuyên gia (cơ sở tri thức, cơ sở luật, động cơ suy diễn, giao diện người dùng) cùng quy trình xây dựng một hệ chuyên gia; các ứng dụng tự động hoá và AI: xe tự lái (camera, LIDAR, cảm biến siêu âm, GPS), drone, thiết bị nhà thông minh/IoT, rô-bốt phẫu thuật, chatbot, nhận diện khuôn mặt; lợi ích, hạn chế và các vấn đề đạo đức — chi phí, mất việc làm, thiên vị dữ liệu (bias) và trách nhiệm an toàn khi hệ thống tự động gây lỗi.

6.1 Robotics and control systems

6.1.1 What is a robot?

A **robot** is a programmable physical machine that can sense conditions in its surroundings and act upon its environment, usually carrying out a physical task that would otherwise be done by a human. What separates a robot from an ordinary machine (such as a washing machine that simply runs one fixed sequence of steps regardless of what is happening around it) is that a robot's behaviour is driven by data it continuously gathers about the real world, so its actions can change if the conditions around it change.

Khái niệm cốt lõi

Every robot combines three physical components, working together in a continuous cycle:

- **sensors** — devices that gather data about a physical quantity in the robot's environment (e.g. light, temperature, distance, moisture, pressure);
- a **processor** (control unit) — runs a stored program that reads the sensor data, compares it against pre-set values or applies a more elaborate decision-making algorithm, and decides what action (if any) is needed;
- **actuators** — physical components (motors, motorised arms, valves, heaters, switches) that carry out the action the processor has decided on, changing something in the real, physical world.

VN **GIẢI THÍCH TIẾNG VIỆT**

Rô-bốt (robot): máy vật lý có thể lập trình, cảm nhận môi trường xung quanh và tác động lên môi trường đó — khác với máy móc thông thường chỉ lặp lại một chuỗi bước cố định. Rô-bốt gồm ba thành phần: **cảm biến (sensors)** thu thập dữ liệu về đại lượng vật lý (ánh sáng, nhiệt độ, khoảng cách, độ ẩm, áp suất); **bộ xử lý (processor)** chạy chương trình so sánh dữ liệu cảm biến với giá trị đặt trước hoặc chạy thuật toán quyết định phức tạp hơn; **cơ cấu chấp hành (actuators)** (động cơ, cánh tay máy, van, bộ sưởi, công tắc) thực hiện hành động mà bộ xử lý đã quyết định, tác động thực sự lên thế giới vật lý.

6.1.2 The sense–think–act feedback loop

The three components above do not act once and stop — they operate together in an endless cycle called a **feedback loop**, because the outcome of one action changes the very conditions that are sensed next, which in turn influences the next decision.

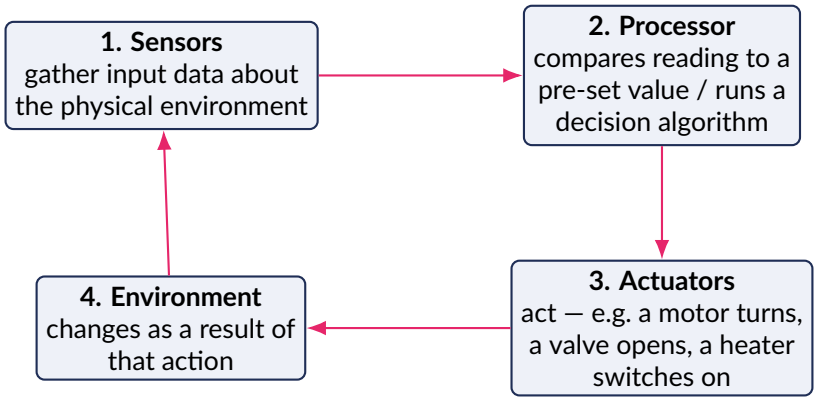


Figure 1. The sense–think–act feedback loop: each stage leads into the next, and the loop repeats continuously without human intervention, so the system keeps adjusting itself as real-world conditions change.

VN **GIẢI THÍCH TIẾNG VIỆT**

Vòng lặp phản hồi (feedback loop): (1) **cảm biến** thu thập dữ liệu đầu vào về môi trường vật lý; (2) **bộ xử lý** so sánh giá trị đọc được với giá trị đặt trước hoặc chạy thuật toán ra quyết định; (3) **cơ cấu chấp hành** thực hiện hành động (động cơ quay, van mở, bộ sưởi bật); (4) **môi trường** thay đổi do kết quả của hành động đó — rồi vòng lặp lại từ đầu, liên tục, không cần con người can thiệp. Gọi là "phản hồi" vì kết quả của hành động trước sẽ ảnh hưởng đến dữ liệu được cảm nhận ở lần tiếp theo.

Ví dụ mẫu · Worked example

A greenhouse contains an automated watering system: a **soil-moisture sensor** buried in the soil, a **processor** running a stored program, and a **solenoid valve** connected to a water pipe as the actuator. The system is programmed to keep the soil between 25 % and 35 % moisture without any human intervention. Describe, stage by stage, how the feedback loop achieves this, and explain why the system uses *two* threshold values rather than one.

Sensing: the moisture sensor repeatedly samples the soil and sends a moisture reading to the processor.
Processing: the processor compares the current reading against the two pre-set values stored in its program — a lower threshold (25 %) and an upper threshold (35 %). If the reading falls to or below 25 %, the processor decides the valve should open; once the reading rises to or above 35 %, it decides the valve should close.
Acting: the actuator (solenoid valve) physically opens or closes as instructed, allowing water to flow onto the soil or stopping it.
Environment changes: the soil's moisture level rises while the valve is open, changing the very quantity the sensor next measures.
Loop repeats: the sensor takes a fresh reading, and the whole cycle runs again, continuously, for as long as the system is powered.

Why two thresholds, not one: if the system used a single exact target (say, exactly 30 %), tiny natural fluctuations in soil moisture right around that value would repeatedly push the reading a fraction above and below it, causing the valve to open and close very rapidly and repeatedly (**chattering**). Using a lower "open" threshold and a separate, higher "close" threshold creates

a **tolerance band**: once the valve opens, it stays open until moisture is clearly well above the target, and once closed, it stays closed until moisture clearly falls again — avoiding rapid, wasteful, mechanically damaging switching.

Ví dụ mẫu · Worked example

A room's heating system uses a **temperature sensor**, a **processor**, and an electric **heater** controlled by a relay (the actuator). The system is programmed to switch the heater **on** whenever the temperature falls to or below 19.5 °C, and **off** whenever it rises to or above 20.5 °C. Trace the feedback loop through one full cycle, starting from a cold room at 18 °C.

Sensing: the temperature sensor reads 18 °C and passes this reading to the processor.
Processing: the processor compares 18 °C against the stored switch-on threshold (19.5 °C). Since 18 °C is below this threshold, it decides the heater should be switched **on**.
Acting: the relay closes the heater's circuit, and the heater begins warming the room.
Environment changes: the room's air temperature rises over time.
Loop repeats: the sensor keeps sampling; once the reading reaches 20.5 °C (the switch-off threshold), the processor decides to switch the heater **off**. The room then cools naturally until the temperature falls back to 19.5 °C, at which point the heater switches on again, and the cycle continues indefinitely, keeping the room's temperature oscillating within the 19.5–20.5 °C band without anyone touching a thermostat dial.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cả hai ví dụ (tưới cây tự động và hệ thống sưởi) đều dùng **hai ngưỡng** (một ngưỡng bật/mở, một ngưỡng tắt/đóng) thay vì một giá trị duy nhất, để tạo ra một **khoảng dung sai (tolerance band/hysteresis)** — tránh hiện tượng cơ cấu chấp hành bật/tắt liên tục quá nhanh (gọi là *chattering*) khi giá trị đo dao động nhẹ quanh một mốc duy nhất.

(!) Lỗi thường gặp: Do not describe a feedback loop as a one-off event ("the sensor detects X, then the actuator does Y, the end"). The defining feature of a feedback *loop* is that it repeats continuously — the environment change caused by the actuator becomes the new input the sensor measures next, which is why the system keeps self-correcting without a person checking on it.

6.1.3 Robots compared with human workers

Industrial robots and human workers each have genuine strengths, and IGCSE-style questions often ask for a balanced comparison rather than simply declaring one "better".

Dimension	Industrial robot	Human worker
Precision & consistency	Repeats an identical action to a very high, unchanging precision, thousands of times without fatigue-related drift.	Precision can vary between individuals and declines with fatigue, distraction or repetitive-strain injury over a long shift.
Speed	Typically performs repetitive physical actions (welding, assembling, packing) far faster than a human, and can run continuously, 24 hours a day, without rest breaks.	Limited by human reaction time and needs regular rest breaks, sleep, and holidays.
Cost	High **upfront** cost to purchase, install and programme; but relatively low **ongoing** cost (electricity and occasional maintenance), so it becomes cheaper over a long enough period of continuous use.	Low upfront cost to hire, but ongoing wages, benefits and training must be paid indefinitely for as long as the person is employed.
Hazardous/extreme environments	Can operate in conditions immediately dangerous or impossible for people — toxic fumes, extreme heat, radiation, deep water, or handling heavy/sharp components.	Exposing a person to the same conditions risks injury, illness or death, and normally requires expensive protective equipment or is simply not permitted.
Flexibility with the unexpected	Only handles situations its sensors and program were designed for; an unforeseen object, fault, or unusual instruction can leave it unable to act sensibly, or unable to act at all.	Can use judgement, creativity and general knowledge to improvise a sensible response to a situation nobody anticipated.
Job/social impact	Automating a task removes the need for the human workers who previously performed it, which can cause job losses and require retraining for other roles.	Employment provides income, skill development and social structure for the people directly involved.

Industrial robots compared with human workers across several practical dimensions.

VN **GIẢI THÍCH TIẾNG VIỆT**

So với người lao động, rô-bốt công nghiệp có **độ chính xác/ổn định cao hơn**, **tốc độ nhanh hơn**, hoạt động được trong **môi trường nguy hiểm** mà không gây rủi ro cho con người, và về lâu dài có **chi phí vận hành thấp hơn** dù chi phí đầu tư ban đầu rất cao. Tuy nhiên rô-bốt **kém linh hoạt** khi gặp tình huống bất ngờ nằm ngoài thiết kế, và việc tự động hoá có thể gây **mất việc làm** cho người lao động từng đảm nhận công việc đó.

(!) Lỗi thường gặp: When a question asks you to "evaluate" or "discuss" automating a task, do not list only benefits or only drawbacks. A strong answer weighs a genuine advantage (e.g. precision, cost over time, safety) against a genuine disadvantage (e.g. upfront cost, job losses, inflexibility) specific to the scenario given, rather than a generic, one-sided list.

6.2 Artificial intelligence

6.2.1 Machine learning

Artificial intelligence (AI) is the broad field concerned with building systems that perform tasks which would normally be described as requiring human intelligence — recognising images, understanding language, making predictions, or reaching decisions. **Machine learning** is one major approach to AI

in which a system improves its performance on a task by learning patterns from **data**, rather than following only rules that a programmer wrote out in advance and fixed for all time.

Khái niệm cốt lõi

A traditional, fixed rule-based program behaves *exactly* as its programmer specified, forever: given the same input, it always produces the same output, and it never changes its own behaviour based on outcomes it observes. A **machine learning** system instead starts from a large set of **training data** (often labelled with the correct answer) and uses that data to build up its own internal model of the patterns that distinguish one category, or one outcome, from another — and it can continue to be refined as *more* data becomes available, improving its accuracy over time without a human having to rewrite its rules by hand.

Example — a spam email filter. A simple rule-based filter might block any email containing a small, fixed list of banned words. A **machine-learning** spam filter instead is trained on thousands of emails already labelled "spam" or "not spam"; it learns which combinations of features (sender patterns, wording, formatting, links) tend to appear in spam, builds a model from these patterns, and can keep improving as it processes more emails and as users mark further messages as spam or not — catching cleverly-worded spam that never appears on any fixed banned-word list.

VN **GIẢI THÍCH TIẾNG VIỆT**

Học máy (machine learning): hệ thống cải thiện hiệu suất qua việc **học các quy luật từ dữ liệu**, thay vì chỉ tuân theo các quy tắc cố định do lập trình viên viết sẵn. Ví dụ bộ lọc thư rác (spam filter) học máy được huấn luyện trên hàng ngàn email đã gán nhãn "spam"/"không spam", tự rút ra các đặc điểm chung của thư rác và tiếp tục cải thiện độ chính xác khi có thêm dữ liệu mới — khác với bộ lọc dựa trên danh sách từ cấm cố định, vốn không bao giờ tự thay đổi.

Ví dụ mẫu · Worked example

Explain, with reference to a spam filter, why a machine-learning approach can catch spam messages that a fixed-rule, banned-word-list approach would miss.

A banned-word-list filter can only block a message if it contains one of the exact words on its fixed list — a spam sender who avoids those specific words, or who deliberately misspells them, passes straight through undetected, and the list only grows if a programmer manually edits it. A **machine-learning** filter, by contrast, has learned broader statistical patterns from a very large set of previously-labelled examples — for instance, unusual sender/formatting combinations, or wording patterns that only loosely resemble known spam — so it can flag a message as likely spam even if it does not contain any specific banned word, and it keeps adapting as new spam patterns appear in the data it continues to process.

6.2.2 Expert systems

An **expert system** is a program designed to imitate the decision-making of a human expert within one narrow, specific field, by storing that expert's knowledge and reasoning as data and rules, and applying it automatically to a new situation the user describes.

Khái niệm cốt lõi

An expert system always has four components:

- a **knowledge base** — a large store of facts about the specific subject area (e.g. known diseases and the symptoms associated with each one);

- a **rules base** — a set of **IF–THEN** rules that encode expert reasoning about how those facts should be combined and interpreted (e.g. "IF fever AND cough AND sore throat THEN suggest influenza");
- an **inference engine** — the software component that compares facts the user has entered against the rules base, and chains rules together logically to reach a conclusion, exactly as a human expert would reason step by step;
- a **user interface** — lets a non-expert user enter information (e.g. answer questions about symptoms) and displays the system's conclusion, often together with an explanation of which rules led to it.

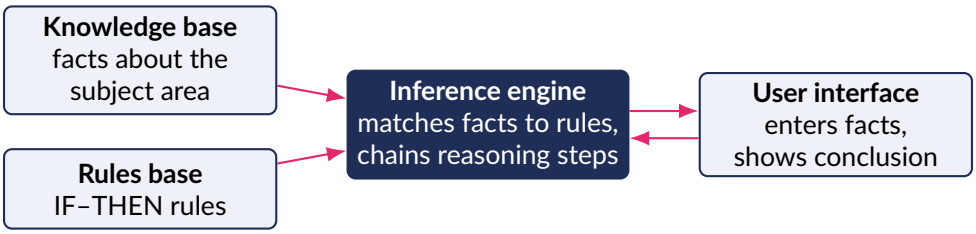


Figure 2. The four components of an expert system: the inference engine draws on the knowledge base and rules base to reason from the facts a user enters (via the user interface) to a conclusion, which is then displayed back to the user.

VN **GIẢI THÍCH TIẾNG VIỆT**

Hệ chuyên gia (expert system) luôn gồm bốn thành phần: **cơ sở tri thức (knowledge base)** — kho dữ kiện về lĩnh vực chuyên môn; **cơ sở luật (rules base)** — tập hợp các luật **IF–THEN** mã hoá cách chuyên gia con người suy luận; **động cơ suy diễn (inference engine)** — so khớp dữ kiện người dùng nhập với các luật, kết nối các luật lại để đi đến kết luận; **giao diện người dùng (user interface)** — cho người dùng không chuyên nhập thông tin và xem kết luận (thường kèm giải thích).

Ví dụ mẫu · Worked example

A patient enters their symptoms — *high temperature*, *persistent cough*, and *loss of taste* — into a medical diagnosis expert system. Explain the role each of the four components plays in producing a suggested diagnosis.

Knowledge base: stores facts linking many known illnesses to the symptoms typically associated with each of them (e.g. that loss of taste is strongly associated with certain viral infections).
Rules base: contains IF–THEN rules such as "IF high temperature AND persistent cough AND loss of taste THEN suggest a particular viral respiratory infection".
Inference engine: takes the three symptoms the patient entered, checks them against the rules base, finds that all three conditions of the matching rule are satisfied, and follows the chain of reasoning that rule specifies to reach the corresponding suggested diagnosis (chaining further rules together if the first conclusion triggers additional questions, e.g. asking about symptom duration).
User interface: presented the original questions to the patient in plain language, collected their answers, and now displays the suggested diagnosis — often alongside an explanation such as "because you reported a high temperature, cough and loss of taste" — while making clear the system's output is a suggestion, not a certified medical diagnosis from a doctor.

6.2.3 Building an expert system

The knowledge base and rules base of an expert system do not appear automatically — they must be carefully extracted from real human expertise before the system can be used.

Khái niệm cốt lõi

A **knowledge engineer** is the person (or team) responsible for building an expert system's knowledge base and rules base. They do this by:

- **interviewing human domain experts** (e.g. experienced doctors, geologists, or loan officers) to draw out the facts they rely on and the reasoning steps they follow when reaching a decision;
- **consulting documented sources** in the field — textbooks, medical guidelines, technical manuals, published research — to gather further established facts and standard procedures;
- **encoding** everything gathered into the precise, structured form the expert system needs: individual facts stored in the knowledge base, and step-by-step reasoning expressed as formal IF–THEN rules in the rules base;
- **testing** the resulting system against cases with a known, correct outcome, and refining the rules with the experts if the system's conclusions do not match what a human expert would have concluded.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kỹ sư tri thức (knowledge engineer) xây dựng hệ chuyên gia bằng cách: **phỏng vấn chuyên gia con người** để rút ra dữ kiện và cách họ suy luận; **tham khảo tài liệu chuyên ngành** (sách giáo khoa, hướng dẫn, nghiên cứu) để bổ sung dữ kiện đã được ghi nhận; **mã hoá** tất cả thành dữ kiện trong cơ sở tri thức và luật IF–THEN hình thức trong cơ sở luật; sau đó **kiểm thử** hệ thống với các trường hợp đã biết kết quả đúng và tinh chỉnh cùng chuyên gia nếu kết luận của hệ thống chưa khớp.

Ví dụ mẫu · Worked example

A bank wants to build an expert system that advises loan officers on whether to recommend a customer's mortgage application for further approval. Describe how a knowledge engineer would build this system, and then explain the role of each of the four expert-system components in this new domain.

Building the system: the knowledge engineer interviews experienced loan officers to find out which factors they weigh (income, credit history, existing debt, deposit size) and how they combine them when deciding whether an application looks safe to recommend; they also consult the bank's official lending policy documents and regulatory guidelines for any fixed rules that must always be applied. All of this is encoded as facts and formal IF–THEN rules, and the resulting system is tested against a sample of past applications with a known outcome before being put into use.

Knowledge base: stores facts such as typical minimum deposit percentages, acceptable debt-to-income ratio ranges, and categories of credit history.
Rules base: contains rules such as "IF deposit $\geq$ 20 % of property value AND debt-to-income ratio $<$ 35 % AND credit history is good THEN recommend approval".
Inference engine: takes the specific figures entered for one applicant, checks them against every relevant rule, and follows the matching rule (or chain of rules, if an initial conclusion triggers a request for further information) to reach a recommendation.
User interface: lets the loan officer enter the applicant's financial details in a simple form, and

displays the system's recommendation together with the reasons behind it, so the officer can review the reasoning before making the final decision themselves.

An expert system's conclusion is only as reliable as the knowledge and rules a knowledge engineer encoded into it. If the underlying facts are out of date, or the rules were built from an unrepresentative set of past cases, the system's recommendations can be systematically wrong even though the inference engine itself is working exactly as designed.

6.3 Applications of automation and AI

6.3.1 Self-driving cars: fusing several sensors into one decision

A self-driving (autonomous) car is a direct, safety-critical extension of the feedback loop introduced earlier in this chapter: it continuously senses its surroundings using *several different kinds of sensor at once*, and its onboard processor must combine (**fuse**) all of that data into a single, continuously-updated driving decision.

Khái niệm cốt lõi

A self-driving car typically combines four main types of sensing technology:

- **Cameras** — capture visual images used to recognise road markings, lane boundaries, traffic signs, traffic lights, and the presence of pedestrians or other vehicles, much as a human driver uses their eyes.
- **LIDAR** (Light Detection And Ranging) — fires rapid pulses of laser light and measures how long they take to reflect back, building a precise, continuously-updated 3-D map of distances to every nearby object, in almost any lighting condition.
- **Ultrasonic sensors** — emit high-frequency sound pulses to detect the distance to *very nearby* objects; short in range but useful for close-quarters tasks such as parking and detecting a kerb.
- **GPS** (Global Positioning System) — receives signals from satellites to determine the car's current location and combines this with stored map data for route planning and navigation.

The onboard processor continuously fuses the input from all four: for example, the camera might identify a shape ahead as a pedestrian, while LIDAR simultaneously confirms its precise distance and that it is moving into the car's path — the processor combines both readings before deciding to brake, exactly as stage 2 ("processor") of the general feedback loop compares sensor data against pre-set conditions before instructing an actuator.

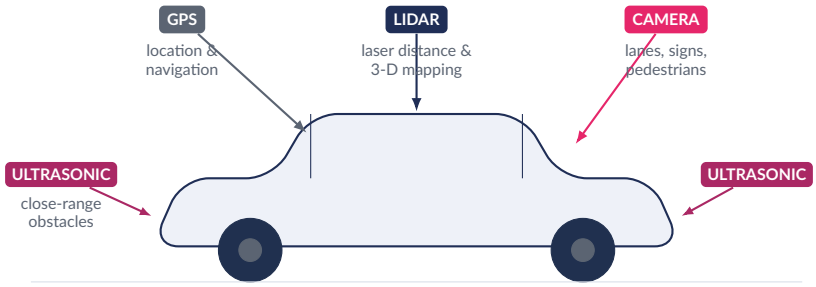


Figure 3. A self-driving car's sensor suite: cameras and LIDAR cover long-range recognition and mapping, ultrasonic sensors cover very short-range obstacles (e.g. parking), and GPS provides location for navigation — the onboard processor fuses all four continuously.

VN **GIẢI THÍCH TIẾNG VIỆT**

Xe tự lái là một ứng dụng mở rộng của vòng lặp phản hồi, kết hợp nhiều loại cảm biến cùng lúc: **camera** (nhận diện vạch kẻ đường, biển báo, người đi bộ), **LIDAR** (đo khoảng cách bằng laser, dựng bản đồ 3 chiều), **cảm biến siêu âm** (phát hiện vật cản ở khoảng cách rất gần, ví dụ khi đỗ xe), và **GPS** (xác định vị trí để dẫn đường). Bộ xử lý trên xe liên tục **kết hợp (fuse)** dữ liệu từ tất cả các cảm biến này để đưa ra quyết định lái xe theo thời gian thực.

Ví dụ mẫu · Worked example

A self-driving car is travelling on a motorway in thick fog, which significantly reduces how far and how clearly its cameras can see. Explain why relying on cameras alone would be dangerous in this situation, and explain how the car's other sensors allow it to keep driving safely.

Cameras rely on **visible light** reflecting off objects into the lens, so thick fog scatters and blocks that light, sharply reducing the distance and clarity at which the camera can reliably recognise lane markings, signs, or other vehicles — if the processor depended on the camera alone, it might fail to detect a hazard until far too late to react safely.

LIDAR is far less affected by fog than a visible-light camera at the ranges relevant to safe braking distance, and continues to provide accurate distance measurements to nearby vehicles and obstacles. **Ultrasonic sensors** continue to detect very close obstacles regardless of fog, since they rely on sound rather than light. **GPS** continues to confirm the car's position on the road and its planned route regardless of visibility. By fusing LIDAR and ultrasonic readings (largely unaffected by fog) with whatever the camera can still detect, and by reducing speed to keep a safe stopping distance, the processor can continue to make safe driving decisions even while one sensor (the camera) is significantly degraded — illustrating why self-driving cars are designed with *several different*, *complementary* sensor types rather than relying on just one.

6.3.2 Drones

A **drone** is an unmanned aerial vehicle that can fly either under direct remote control by a human operator, or autonomously, using GPS for navigation together with onboard sensors (cameras, altitude and proximity sensors) to follow a planned route and avoid obstacles without a person steering it moment to moment.

VN **GIẢI THÍCH TIẾNG VIỆT**

Drone (thiết bị bay không người lái): có thể bay theo điều khiển từ xa của con người, hoặc **tự động** bằng GPS kết hợp cảm biến trên khoang (camera, cảm biến độ cao, cảm biến vật cản) để bay theo lộ trình đã định và tự tránh vật cản.

Common uses include **aerial photography and filming** (capturing footage from viewpoints impossible or dangerous for a person to reach), **package delivery** in areas difficult to reach by road, and **agricultural monitoring** (flying over large fields to photograph crop health, moisture patterns, or pest damage far faster than a person could walk the same area).

6.3.3 Smart home and Internet of Things (IoT) devices

The **Internet of Things (IoT)** refers to everyday physical devices — fitted with sensors, actuators and network connectivity — that can gather data, communicate with each other, and be controlled remotely over a network, typically the home's own Wi-Fi network and, through it, the wider internet.

Khái niệm cốt lõi

Common **smart home** devices include: a **smart thermostat** (senses room temperature and can be programmed or learn a household's schedule to automatically heat or cool the home efficiently); **smart security cameras** (sense motion and stream or record video, sending an alert to the owner's phone over the network); and **voice assistants** (listen for spoken commands and act as a hub instructing other connected devices — lights, locks, thermostats — what to do). All of these devices apply the same underlying sense–think–act feedback loop described earlier in this chapter, but they additionally communicate their data, and receive instructions, over a **home network** rather than acting in isolation.

VN **GIẢI THÍCH TIẾNG VIỆT**

Internet of Things (IoT): các thiết bị vật lý hằng ngày được gắn cảm biến, cơ cấu chấp hành và kết nối mạng, có thể thu thập dữ liệu, giao tiếp với nhau và được điều khiển từ xa qua mạng gia đình/Internet. Ví dụ: **bộ điều nhiệt thông minh**, **camera an ninh thông minh**, **trợ lý giọng nói** — tất cả đều áp dụng vòng lặp phản hồi cảm biến-xử lý-chấp hành, nhưng còn giao tiếp dữ liệu qua mạng gia đình.

Ví dụ mẫu · Worked example

A smart thermostat learns that a family typically leaves home at 8 a.m. and returns at 6 p.m. each weekday, and automatically lowers the heating while the house is empty, raising it again shortly before the family is expected home. Identify which technology from this chapter allows the thermostat to "learn" this schedule rather than simply follow a fixed timer, and explain the difference.

This is an application of **machine learning**. A fixed timer would need a person to manually programme the exact departure and return times, and would never adjust itself if the family's routine changed. A machine-learning thermostat instead observes the pattern of when the house is occupied (from motion sensors, connected phones, or manual adjustments logged over many days) and builds its own model of the household's typical schedule from that data, automatically refining its heating schedule as the observed pattern is confirmed or as it changes over time — without a person ever re-programming a fixed timer.

6.3.4 Further applications: robots, chatbots and facial recognition

Automation and AI now appear across many other everyday and industrial settings. **Industrial robots** on a production line perform precise, repetitive tasks such as welding or assembling components at high speed. **Surgical robots** let a trained surgeon control extremely precise, steady instruments (often filtering out any natural hand tremor) during delicate procedures. **Chatbots** use AI, often including machine learning over large amounts of past conversation data, to hold a text or voice conversation with a user and answer routine queries without a human member of staff being involved for every request. **Facial recognition** systems use AI to analyse a photograph or video of a face and match it against a stored database, used for tasks such as unlocking a phone or identifying a person passing through a security checkpoint.

6.4 Benefits, drawbacks and ethical considerations

6.4.1 Benefits and drawbacks of automation and AI

Aspect	Benefit	Drawback
Speed & precision	Automated systems perform repetitive physical or computational tasks faster, and to a more consistent standard, than a human typically can.	Precision is only as good as the sensors, program and training data used — a poorly calibrated sensor or an unrepresentative training set can make an automated system confidently wrong.
Availability	Can run continuously, 24 hours a day, 7 days a week, without needing rest, unlike a human employee.	Requires reliable power and network connectivity; a fault, power cut or network outage can halt operation entirely, with no human fallback immediately available.
Cost	Long-term running costs (electricity, occasional maintenance) can be lower than paying wages indefinitely, once the initial investment is recovered.	High **initial setup cost** to purchase, install, programme and test the system, which can be a substantial barrier, especially for smaller organisations.
Safety	Removes people from immediately hazardous environments (toxic, extreme-temperature, or high-radiation settings).	An automated system malfunctioning in a hazardous environment can itself create a new safety incident, and may be harder to diagnose remotely than a human simply reporting a problem.
Employment	Frees human workers from repetitive, physically demanding tasks, potentially redirecting them to more skilled roles.	Job losses occur for workers whose previous role has been automated, and retraining is not always fast, affordable, or guaranteed to succeed.
Handling the unexpected	A well-designed system can react to a sensed condition far faster than a human, within the range of situations it was designed for.	Automated and AI systems can struggle badly, or fail unsafely, when a genuinely unexpected situation arises that lies outside their training data or programmed rules.

Benefits and drawbacks of automation and AI, considered side by side.

VN **GIẢI THÍCH TIẾNG VIỆT**

Lợi ích của tự động hoá/AI: **tốc độ và độ chính xác** cao hơn, hoạt động **24/7**, chi phí vận hành lâu dài thấp hơn, an toàn hơn trong môi trường nguy hiểm. Hạn chế: **chi phí đầu tư ban đầu** cao, có thể gây **mất việc làm**, độ chính xác phụ thuộc vào chất lượng cảm biến/dữ liệu huấn luyện, và hệ thống có thể **xử lý kém** khi gặp tình huống bất ngờ nằm ngoài thiết kế hoặc dữ liệu huấn luyện.

6.4.2 Bias in AI systems

An AI system trained on data can only learn the patterns that are actually present in the data it was given. If that training data does not fairly represent every group of people the system will later be used on, the system's decisions can become systematically less accurate, or less fair, for the groups that were underrepresented — this is called **bias**.

Ví dụ mẫu · Worked example

A facial recognition system used at airport security gates is found to misidentify passengers with darker skin tones considerably more often than passengers with lighter skin tones. Explain the likely underlying cause of this problem, and suggest one concrete way the developers could reduce it.

Likely cause: the system's underlying machine-learning model was almost certainly trained on

a dataset of face images that contained far more examples of lighter-skinned faces than darker-skinned faces. Because the model learns its recognition patterns purely from the examples it was shown, it develops a less accurate internal model for the kinds of faces it saw *fewer* labelled examples of, leading to a systematically higher error rate for that group — this is not a fault in the general *idea* of facial recognition, but a direct consequence of an unrepresentative training dataset.

Mitigation: the developers should deliberately collect and add a much larger, more balanced set of labelled training images that fairly represents the full range of skin tones (and other relevant variation, such as age and gender) present among the people the system will actually be used on, then retrain and re-test the model's accuracy separately for each group before deployment, continuing to monitor real-world accuracy across groups after release rather than relying only on one overall accuracy figure that could hide an unfair gap between groups.

Bias in an AI system is a problem with the **data it learned from** (or, occasionally, the assumptions built into its rules), not a problem inherent to the general idea of using AI. Two systems trained on the same technology can be much more or much less biased, purely depending on how representative their training data was.

6.4.3 Safety, testing and accountability for autonomous systems

Because an autonomous system such as a self-driving car makes real-time physical decisions that can directly affect human safety, its development and deployment raises questions that go beyond ordinary software testing.

Khái niệm cốt lõi

Before an autonomous system is deployed in situations that affect public safety, it is typically subjected to **extensive testing** — across a very wide range of conditions (weather, lighting, unusual obstacles, sensor faults) and often over millions of simulated and real test kilometres or hours — precisely *because* a fault discovered only after deployment could cause physical harm rather than merely an inconvenient software bug. Even after extensive testing, a genuinely novel situation not covered during testing can still arise once the system is in widespread real-world use.

When an automated or AI-driven system makes an error that causes harm, questions of **accountability** arise: who is responsible when a self-driving car's software misjudges a hazard — the manufacturer that designed and tested the system, the owner/operator of the vehicle at the time, or some shared combination of both? Different jurisdictions and organisations answer this question differently, and it remains an actively-discussed issue as autonomous systems become more widespread; what is broadly agreed is that thorough testing before deployment, together with clear logging of the system's sensor data and decisions (so an incident can later be investigated), are both essential regardless of how accountability is ultimately assigned.

VN **GIẢI THÍCH TIẾNG VIỆT**

Do hệ thống tự động (như xe tự lái) đưa ra quyết định vật lý ảnh hưởng trực tiếp đến an toàn con người, chúng cần được **kiểm thử rất kỹ lưỡng** trước khi triển khai, qua nhiều điều kiện khác nhau (thời tiết, ánh sáng, vật cản bất thường, lỗi cảm biến). Khi hệ thống tự động gây ra lỗi dẫn đến thiệt hại, câu hỏi về **trách nhiệm** được đặt ra — nhà sản xuất, người vận hành, hay cả hai — và đây vẫn là vấn đề đang được thảo luận khi công nghệ tự động ngày càng phổ biến.

Dù trách nhiệm được quy định thế nào, việc kiểm thử kỹ trước khi triển khai và ghi log đầy đủ dữ liệu cảm biến/quyết định của hệ thống đều được xem là cần thiết để có thể điều tra sự cố sau này.

6.5 Practice

Bài tập luyện tập

A streetlight automatically switches on at dusk and off at dawn using a light sensor. Which stage of the feedback loop does the light sensor represent?

(A) Actuator
(B) Processor
(C) Sensor
(D) Environment

Lời giải chi tiết

The light sensor is the component that **gathers input data** about a physical quantity (ambient light level) — this is the **sensor** stage of the feedback loop. **(C)**.

Bài tập luyện tập

Put the following stages of a general feedback loop into the correct order: (i) the actuator acts, (ii) the environment changes, (iii) sensors gather input data, (iv) the processor compares the reading to a pre-set value or runs a decision algorithm.

Lời giải chi tiết

Correct order: **(iii)** sensors gather input data → **(iv)** the processor compares the reading to a pre-set value / runs a decision algorithm → **(i)** the actuator acts → **(ii)** the environment changes — after which the loop repeats from stage (iii) again, since the changed environment produces new sensor data.

Bài tập luyện tập

In an expert system, which component is responsible for matching the facts a user has entered against the rules base and chaining reasoning steps together to reach a conclusion?

(A) Knowledge base
(B) User interface
(C) Inference engine
(D) Rules base

Lời giải chi tiết

The **inference engine** performs the actual reasoning: it compares entered facts against the rules base and chains matching rules together to reach a conclusion. **(C)**.

Bài tập luyện tập

A car manufacturer replaces a team of workers who weld car body panels by hand with a robotic welding arm that repeats the same weld with extremely high precision, 24 hours a day. State one benefit and one drawback of this change.

Lời giải chi tiết

Benefit: the robotic arm performs each weld with much greater precision and consistency than a human can sustain over a long shift, and can operate continuously without rest breaks, increasing output.

Drawback: the workers who previously did this welding lose their jobs, and the manufacturer must pay a large upfront cost to purchase, install and programme the robotic arm before it produces any output at all.

Bài tập luyện tập

A facial recognition system trained mostly on photographs of adult faces performs noticeably worse at recognising children's faces. Explain the likely cause of this bias, and suggest a way to reduce it.

Lời giải chi tiết

Likely cause: the system's underlying model was trained on a dataset containing far fewer labelled examples of children's faces than adult faces, so it learned a less accurate internal model for recognising the facial patterns and proportions typical of children.

Mitigation: developers should add a substantial, properly-labelled set of children's face images to the training data so that both groups are fairly represented, then retrain the model and test its accuracy separately for children and adults before it is deployed, rather than relying on one combined accuracy figure that could hide the gap between the two groups.

Bài tập luyện tập

Which of the following best distinguishes a **robot** from an ordinary, non-programmable machine such as a basic wind-up toy?

(A) A robot is always larger than an ordinary machine

(B) A robot's behaviour is driven by continuously-sensed data, allowing it to change if conditions change, rather than repeating one fixed sequence regardless of its surroundings

(C) A robot never requires any actuators

(D) A robot cannot be used in industry

Lời giải chi tiết

A robot combines sensors, a processor and actuators so that its actions respond to continuously-sensed real-world conditions, unlike a simple machine that runs one fixed sequence of steps no matter what is happening around it. **(B)**.

Bài tập luyện tập

An automatic room heating system switches the heater on at 19.5 °C and off at 20.5 °C, rather than using a single target of exactly 20 °C. Explain why using two separate threshold values is better than using one.

Lời giải chi tiết

If a single exact threshold (20 °C) were used, small natural fluctuations in room temperature right around that value would repeatedly push readings a fraction above and below it, causing

the heater to switch on and off very rapidly and repeatedly (chattering), which wastes energy and wears out the switching mechanism. Using a lower "switch-on" threshold and a separate, higher "switch-off" threshold creates a tolerance band: once the heater switches on, it stays on until the temperature clearly rises well above the target, and once off, it stays off until the temperature clearly falls again — avoiding rapid, repeated switching.

Bài tập luyện tập

According to the comparison between industrial robots and human workers, which dimension most clearly favours a human worker over a robot?

(A) Precision and consistency over a long shift

(B) Ability to work continuously without rest

(C) Flexibility to sensibly handle a completely unforeseen situation

(D) Ability to operate in a toxic or extremely hot environment

Lời giải chi tiết

A human can use judgement and general knowledge to improvise a sensible response to a situation nobody anticipated, whereas a robot can only act within the range of situations its sensors and program were designed to handle. **(C)**.

Bài tập luyện tập

An automatic car park barrier uses a vehicle-detection sensor at the entrance, a processor, and a motorised barrier arm as the actuator. Describe, stage by stage, how the feedback loop allows a car to enter without a human attendant, and explain what causes the loop to repeat for the next car.

Lời giải chi tiết

Sensing: the vehicle-detection sensor at the entrance detects that a car has arrived and sends this reading to the processor.
Processing: the processor checks the reading against its stored condition (e.g. "a vehicle is present and has a valid ticket/permit") and, if satisfied, decides the barrier should be raised.
Acting: the actuator (motorised barrier arm) raises, allowing the car to drive through.
Environment changes: the car passes through and moves beyond the sensor's detection zone, so the sensor no longer detects a vehicle present.
Loop repeats: once the sensor detects the vehicle has cleared the area (or after a short timer), the processor lowers the barrier again; the sensor then continues monitoring, ready to detect the next car and repeat the entire cycle, all without a human attendant present.

Bài tập luyện tập

Which statement correctly distinguishes a machine-learning system from a traditional, fixed rule-based program?

(A) A machine-learning system always runs faster than a rule-based program

(B) A machine-learning system improves its performance by learning patterns from data, whereas a fixed rule-based program always follows only the rules its programmer wrote in advance

(C) A fixed rule-based program can update its own rules automatically, but a machine-learning system cannot

(D) There is no meaningful difference between the two

Lời giải chi tiết

A machine-learning system builds its own internal model of patterns from training data and can continue improving as more data becomes available, while a fixed rule-based program behaves exactly as originally specified and never changes its own behaviour based on outcomes it observes. **(B)**.

Bài tập luyện tập

Explain why a machine-learning spam filter can continue to improve over time, whereas a fixed banned-word-list spam filter cannot, unless a person manually edits it.

Lời giải chi tiết

A fixed banned-word-list filter's behaviour is entirely determined by the specific words on its list; it will only ever block a message containing one of those exact words, and its behaviour never changes unless a programmer manually adds or removes words from the list. A machine-learning spam filter instead builds an internal statistical model from a large set of labelled training emails, and this model can be refined further as *more* labelled emails (including new spam patterns and user "mark as spam" actions) become available — so its accuracy can keep improving automatically over time, without a human rewriting any fixed rules.

Bài tập luyện tập

Which two components together allow an expert system's inference engine to reach a conclusion?

(A) The user interface and the processor

(B) The knowledge base and the rules base

(C) The actuator and the sensor

(D) The training data and the camera

Lời giải chi tiết

The inference engine reasons by matching facts from the **knowledge base** against the **rules base**'s IF–THEN rules, chaining matching rules together to reach a conclusion. **(B)**.

Bài tập luyện tập

Explain the role of a knowledge engineer in building an expert system, and explain why simply asking one expert a single question is usually not sufficient.

Lời giải chi tiết

A **knowledge engineer** builds an expert system's knowledge base and rules base by interviewing human domain experts to extract the facts they rely on and the reasoning steps they follow, and by consulting documented sources (textbooks, guidelines, research) for further established facts, before encoding everything as structured facts and formal IF–THEN rules.

A single question to one expert is usually not sufficient because expert reasoning typically involves *many* interacting facts and conditional rules built up over years of experience, and different experts may emphasise slightly different factors; the knowledge engineer must draw out this reasoning across many interview sessions (and cross-check it against documented sources and, later, real test cases) to build a rules base thorough and reliable enough to give sensible conclusions across the full range of situations the system will actually encounter.

Bài tập luyện tập

An expert system is built to help identify unknown rock and mineral samples from properties such as hardness, colour, lustre and crystal structure. Identify what would form the **knowledge base** and what would form the **rules base** in this system.

Lời giải chi tiết

The **knowledge base** would store facts about known minerals — for example, the typical hardness (on the Mohs scale), colour range, lustre, and crystal structure associated with each named mineral.

The **rules base** would contain IF–THEN rules combining these properties into identification logic — for example, "IF hardness is approximately 7 AND lustre is glassy AND crystal structure is hexagonal THEN suggest quartz" — which the inference engine applies to the specific properties a user has measured and entered for their unknown sample.

Bài tập luyện tập

Which sensor on a self-driving car is primarily responsible for building a precise, continuously-updated 3-D map of distances to nearby objects using reflected laser pulses?

(A) Camera
(B) GPS
(C) LIDAR
(D) Ultrasonic sensor

Lời giải chi tiết

LIDAR (Light Detection And Ranging) fires rapid laser pulses and measures their reflection time to build a precise 3-D map of distances to surrounding objects. **(C)**.

Bài tập luyện tập

Explain why a self-driving car uses ultrasonic sensors *in addition to* LIDAR, rather than relying on LIDAR alone for every distance-sensing task.

Lời giải chi tiết

LIDAR is designed to build a detailed map of the car's surroundings over a relatively long range, which is essential for planning safe driving decisions at normal road speeds. Ultrasonic sensors are specifically suited to **very short-range** detection — for example, judging the precise

remaining gap to a kerb or another vehicle while parking, or detecting an object immediately alongside the car. Using dedicated short-range ultrasonic sensors for these close-quarters tasks, alongside LIDAR's longer-range mapping, gives the car reliable, purpose-suited sensing across the full range of distances it needs to handle, rather than relying on one sensor type to do a job it was not primarily designed for.

Bài tập luyện tập

A self-driving car's camera detects a shape ahead that could be a pedestrian, while its LIDAR simultaneously reports that an object at that exact position is 8 metres away and moving into the car's path. Explain what the processor does with this combined information, and why using both sensors together is safer than using either alone.

Lời giải chi tiết

The processor **fuses** the two readings: the camera's visual classification ("this shape is likely a pedestrian") is combined with LIDAR's precise distance and motion measurement (8 metres away, moving into the car's path) to reach a single, more confident decision — for example, to brake immediately. Using both together is safer than either alone because the camera is better suited to *recognising what* an object is, while LIDAR is better suited to measuring *exactly how far away* it is and how it is moving; combining both readings gives a more complete and more reliable picture than either sensor could provide by itself, reducing the chance of a wrong decision caused by one sensor's individual limitation (e.g. a camera struggling in poor light, or LIDAR alone being unable to classify what kind of object it has detected).

Bài tập luyện tập

Which of the following is a typical use of drones, as distinct from a typical use of ground-based industrial robots?

(A) Welding car body panels on a factory production line

(B) Capturing aerial photography and monitoring large agricultural fields from the air

(C) Performing precise incisions during surgery

(D) Filtering spam email

Lời giải chi tiết

Drones are unmanned aerial vehicles, well suited to tasks that benefit from an aerial viewpoint — aerial photography/filming, package delivery to hard-to-reach areas, and monitoring large agricultural fields far faster than walking them on foot. **(B)**.

Bài tập luyện tập

A smart home security camera detects motion at the front door at night and sends an alert with a short video clip to the owner's phone over the home's Wi-Fi network. Explain how this is an example of the general feedback loop from this chapter, and identify what makes it specifically an "IoT" device rather than a simple standalone sensor.

Lời giải chi tiết

This follows the general feedback loop: the camera's motion sensor **senses** movement (input data); the device's processor compares this against its stored condition ("motion detected while armed") and decides an alert is warranted; the actuator-like action is triggering the recording/alert function; and the environment (a notification now waiting on the owner's phone) changes as a result.

What makes it an **IoT** device, specifically, is that it **communicates its data over a network** (the home Wi-Fi network, and through it, the internet) to another device (the owner's phone) rather than simply acting in isolation — a simple standalone motion sensor might, for example, only sound a local alarm, with no network connectivity and no remote notification.

Bài tập luyện tập

Which of the following is a genuine **drawback** of automating a hazardous industrial task with a robot, rather than a disguised benefit?

(A) The robot can operate continuously without needing rest

(B) The robot requires a large upfront cost to purchase, install and programme before it produces any output

(C) The robot removes people from a dangerous environment

(D) The robot performs the task with very high precision

Lời giải chi tiết

The other three options are genuine benefits of automating the task. The large **upfront cost** to purchase, install and programme the robot is a genuine drawback, since it represents a substantial investment that must be recovered before the automation becomes cost-effective. **(B)**.

Bài tập luyện tập

A self-driving car's software misjudges a hazard in an unusual weather condition that was not represented in its test data, resulting in a collision. Discuss why extensive pre-deployment testing is considered essential for autonomous systems, and identify why the question of accountability in this scenario is not always straightforward.

Lời giải chi tiết

Why extensive testing is essential: unlike an ordinary software bug, which might only cause an inconvenience (e.g. a crashed app), a fault in an autonomous vehicle's software can directly cause physical harm, since the system makes real-time driving decisions with no person available to intervene the instant something goes wrong. Testing across a very wide range of conditions (different weather, lighting, obstacles and sensor faults), often over very large numbers of simulated and real test kilometres, is intended to discover as many potential failure situations as possible *before* the system is used by the public, precisely because the cost of a fault discovered only after deployment can be a real safety incident rather than a minor inconvenience.

Why accountability is not straightforward: the collision arose from a genuinely novel condition that was not represented in the test data, meaning the manufacturer could argue the fault reflects the inherent limits of what testing can cover in advance, while others could argue the

manufacturer should have anticipated and tested for a wider range of weather conditions before deployment. Different parties — the manufacturer that designed and tested the system, and the vehicle's owner/operator at the time — may each bear some responsibility, and there is no single, universally agreed rule for dividing that responsibility; what is generally agreed is the importance of thorough pre-deployment testing and clear logging of sensor data and decisions, so that an incident like this can be properly investigated afterwards.

Bài tập luyện tập

Bias in a machine-learning system is best described as arising from which of the following?

(A) A fault in the programming language used to build the system

(B) Training data that does not fairly represent every group the system will be used on

(C) The system running too slowly

(D) The system being too expensive to install

Lời giải chi tiết

An AI system can only learn the patterns present in the data it was trained on; if that data underrepresents some group of people, the system's accuracy for that group can be systematically worse, which is the underlying cause of bias. **(B)**.

Bài tập luyện tập

A hospital is deciding whether to introduce a surgical robot, controlled by a trained surgeon, for a specific delicate procedure currently performed entirely by hand. Discuss one benefit and one drawback of this proposal, referring specifically to the nature of surgery.

Lời giải chi tiết

Benefit: a surgical robot's actuators can filter out the natural small tremor present in any human hand and translate the surgeon's movements into extremely precise, steady instrument motion, which can improve accuracy during delicate steps of the procedure that require very fine, stable control.

Drawback: introducing the robot requires a very high upfront cost (the robotic system itself, installation, and specific training for the surgical team), and surgeons must gain substantial experience with the new system's controls before operating it on real patients with full confidence — during this transition, the hospital must weigh the precision benefit against the cost and the time needed to build that expertise safely.

Bài tập luyện tập

Explain the difference between the general benefit "an automated system can run 24 hours a day" and the general drawback "an automated system can struggle to handle an unexpected situation", using a self-driving car as your example for both.

Lời giải chi tiết

24-hour operation: a self-driving car's sensors, processor and actuators do not need sleep or rest breaks, so — given sufficient power/fuel and normal operating conditions — it can, in principle, continue driving (or be available to drive) at any hour, unlike a human driver who

becomes fatigued and legally/safely needs rest.

Struggling with the unexpected: the same car's decision-making is ultimately bounded by the range of situations its training data and programmed rules anticipated; if it meets a genuinely novel situation — for example, an unusual obstacle, a rare combination of sensor readings, or a scenario no test route ever included — it may make a poor decision or fail to act safely, whereas a human driver could use judgement to improvise a reasonable response even to a situation they had never encountered before. The two properties are not contradictory: the same system can be tirelessly available around the clock *and* still be limited in how well it copes with a truly unforeseen circumstance.

Bài tập luyện tập

A city's traffic control authority wants to introduce an AI-based traffic-light system that adjusts light timings automatically based on live traffic camera and sensor data, replacing fixed pre-programmed light timings. Identify one likely benefit and one likely risk of this change, and explain what kind of testing would be important before switching every junction in the city over to the new system at once.

Lời giải chi tiết

Likely benefit: the system can continuously sense actual traffic conditions (queue lengths, vehicle counts) at each junction and adjust light timings accordingly in real time, potentially reducing congestion far more effectively than fixed timings that cannot respond to unusual traffic patterns (e.g. an accident on a nearby road diverting extra traffic through the junction).

Likely risk: the system's decisions are only as good as its sensors and its underlying decision algorithm; a sensor fault, an unusual combination of traffic conditions not represented in its testing, or a processing delay could cause it to set unsafe or badly congested timings at a junction, with real safety consequences given that traffic lights directly affect vehicle and pedestrian safety.

Testing recommendation: before a city-wide rollout, the system should be trialled at a small number of representative junctions first, across a wide range of real traffic conditions and over an extended period, with a fallback to the previous fixed-timing behaviour available if a fault is detected — allowing problems to be found and corrected on a limited scale before the system is trusted to control every junction in the city simultaneously.

Algorithm Design and Problem-Solving

Mục tiêu Unit: Phân rã bài toán (decomposition) và trừu tượng hoá (abstraction); mô hình Input–Process–Output (IPO); lưu đồ thuật toán (flowchart) và mã giả (pseudocode) theo chuẩn Cambridge; bảng theo dõi biến (trace table), kể cả với mảng và vòng lặp lồng nhau; thuật toán tìm kiếm tuần tự (linear search) và tìm kiếm nhị phân (binary search); thuật toán sắp xếp nổi bọt (bubble sort) và sắp xếp chèn (insertion sort); kiểm tra hợp lệ dữ liệu (validation: range/length/type/presence/format/check digit) và xác minh dữ liệu (verification); và thiết kế dữ liệu kiểm thử (normal/boundary/erroneous data).

7.1 Problem decomposition and abstraction

Before any algorithm can be designed, a large, real-world problem must be broken down into pieces that are small enough to solve, code and test individually. Two related skills make this possible.

Khái niệm cốt lõi

Decomposition is the process of breaking a large, complex problem down into a set of smaller, more manageable sub-problems, each of which can be tackled (and later combined) far more easily than the original problem as a whole. A large problem is decomposed into sub-problems, and each sub-problem may itself be decomposed further, until every piece is small enough to design an algorithm for directly.

Abstraction is the process of deciding which details of a problem are *relevant* to the task at hand and which can safely be ignored, so that a solution focuses only on what actually matters. Abstraction removes unnecessary complexity (irrelevant real-world detail) while preserving everything the algorithm actually needs.

VN **GIẢI THÍCH TIẾNG VIỆT**

Phân rã (decomposition): chia một bài toán lớn, phức tạp thành nhiều bài toán con nhỏ hơn, dễ giải quyết hơn; mỗi bài toán con lại có thể được phân rã tiếp cho đến khi đủ nhỏ để viết thuật toán trực tiếp. **Trừu tượng hoá (abstraction)**: quyết định chi tiết nào của bài toán thực sự cần thiết cho lời giải và bỏ qua những chi tiết không liên quan — giúp lời giải tập trung đúng vào phần cốt lõi thay vì bị rối bởi độ phức tạp của thế giới thực.

7.1.1 Worked example: decomposing a library loan system

Consider the task “write a program that manages a school library’s book loans”. Stated like this, the problem is far too large to design a single algorithm for. Decomposition splits it into independent sub-tasks, each of which can be designed, coded and tested on its own:

Top-down structure diagram: decomposing the library loan system into independent sub-tasks.

Ví dụ mẫu · Worked example

For the sub-task **"Issue a loan to a member"**, explain (a) one detail that must be kept (relevant to the sub-task) and (b) one real-world detail that abstraction allows the algorithm to ignore.

(a) **Relevant detail to keep:** the algorithm must check whether the requested book is currently available (not already on loan) and whether the member's account is in good standing (e.g. has not exceeded the maximum number of simultaneous loans) — both directly affect whether the loan can legally be issued.

(b) **Irrelevant detail to ignore:** the physical colour of the book's cover, which shelf it happens to sit on, or the member's home address are all real attributes of the situation, but none of them affect whether a loan can be issued — abstraction lets the algorithm designer leave them out entirely.

Decomposition and abstraction work together at every stage of algorithm design: decomposition decides *how the problem is split up*, while abstraction decides, within each resulting sub-problem, *which details the algorithm actually needs to consider*. Skipping decomposition on a large problem makes it too complex to design correctly in one attempt; skipping abstraction results in an algorithm cluttered with irrelevant checks and data that make it harder to write, read and test.

7.2 The Input--Process--Output (IPO) model

Once a problem (or one of its decomposed sub-problems) is clearly defined, it is useful to describe it in terms of exactly three things: what data must be supplied to the algorithm, what happens to that data, and what result comes out.

Khái niệm cốt lõi

The **IPO (Input–Process–Output) model** frames every algorithm as three stages:

- **Input** — the data items the algorithm needs to be given before it can run.
- **Process** — the steps performed on that input data (calculations, comparisons, repetition) to transform it into the required result.
- **Output** — the result(s) produced and communicated back to the user once processing is complete.

Identifying the inputs, processes and outputs *before* writing pseudocode or a flowchart forces the designer to be precise about what the algorithm actually needs and what it must ultimately produce.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mô hình **IPO** mô tả mọi thuật toán qua ba giai đoạn: **Input** (dữ liệu đầu vào cần cung cấp), **Process** (các bước xử lý dữ liệu đó — tính toán, so sánh, lặp lại), và **Output** (kết quả trả về cho người dùng). Xác định rõ ba yếu tố này trước khi viết mã giả hay vẽ lưu đồ giúp người thiết kế thuật toán hiểu chính xác thuật toán cần gì và phải tạo ra kết quả gì.

Ví dụ mẫu · Worked example

Apply the IPO model to a program that calculates a person's Body Mass Index (BMI), where $\text{BMI} = \dfrac{\text{weight (kg)}}{\text{height (m)}^2}$.

Stage	Detail
Input	Weight in kilograms; height in metres.
Process	Square the height; divide the weight by that squared value to compute BMI.
Output	The calculated BMI value (optionally, a category such as "healthy weight" derived from it).

IPO breakdown of a BMI calculator.

7.3 Pseudocode and flowcharts

Once a sub-problem's inputs, processes and outputs are known, the algorithm itself is designed using one of two standard notations: **pseudocode** (structured English-like statements) or a **flowchart** (a diagram of shapes and arrows). Both describe exactly the same logic; the choice between them is usually a matter of preference or which best communicates the design to another programmer.

Cambridge pseudocode conventions used throughout this book: assignment is written <–; input/output are written `INPUT` and `OUTPUT`; selection is `IF...THEN...ELSE...ENDIF` or `CASE OF...ENDCASE`; count-controlled repetition is `FOR...TO...NEXT`; condition-controlled repetition is `WHILE...DO...ENDWHILE` (tests *before* the loop body, so the body may run zero times) or `REPEAT...UNTIL` (tests *after* the loop body, so the body always runs at least once).

Khái niệm cốt lõi

A flowchart uses a small, fixed set of shapes, each with one specific meaning:

- **Oval/rounded terminator** — marks the **start** or **stop** of the algorithm; there is exactly one start and at least one stop shape.
- **Parallelogram** — an **input or output** operation (data entering or leaving the algorithm).
- **Rectangle** — a **process** step: an assignment, calculation, or other action that does not itself involve a decision.
- **Diamond** — a **decision**: a condition that is tested, with the flow of control branching along one of (usually two) labelled arrows depending on the result.

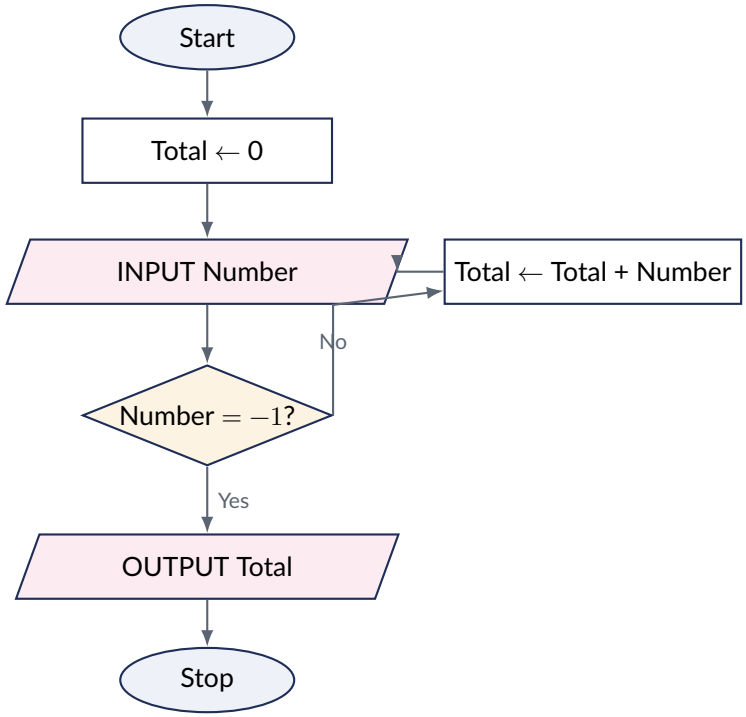
Arrows show the direction of program flow; a diamond is the *only* shape from which more than one arrow may leave.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ý nghĩa các ký hiệu lưu đồ: hình **oval** = bắt đầu/kết thúc (start/stop); hình **bình hành (parallelogram)** = nhập/xuất dữ liệu (input/output); hình **chữ nhật** = một bước xử lý (process, ví dụ phép gán hay tính toán); hình **thoi (diamond)** = một quyết định (decision) — nơi luồng điều khiển rẽ theo một trong các nhánh tuỳ kết quả kiểm tra điều kiện. Chỉ có hình thoi mới có nhiều hơn một mũi tên đi ra.

7.3.1 Worked example: summing numbers until a sentinel value

A common task is to let a user enter as many numbers as they like, adding each one to a running total, and to stop only when a special **sentinel value** (here, -1, which is not a genuine number to be summed) is entered.



Flowchart: sum numbers entered by the user until the sentinel value -1 is entered.

Ví dụ mẫu · Worked example

Write pseudocode matching the flowchart above.

```
Total <- 0
INPUT Number
WHILE Number <> -1 DO
   Total <- Total + Number
   INPUT Number
ENDWHILE
OUTPUT Total
```

The `WHILE` loop tests the sentinel condition *before* each pass, so if the very first number entered is -1, the loop body never executes and `Total` correctly stays at 0.

VN GIẢI THÍCH TIẾNG VIỆT

Giá trị lính canh (sentinel value) là một giá trị đặc biệt (ở đây là -1) không phải là dữ liệu thật cần xử lý, mà chỉ dùng để báo hiệu "dừng nhập dữ liệu". Vòng lặp `WHILE...DO...ENDWHILE` kiểm tra điều kiện **trước** mỗi lần lặp, nên nếu người dùng nhập -1 ngay từ đầu, phần thân vòng lặp không chạy lần nào và `Total` vẫn đúng bằng 0.

(!) Lỗi thường gặp: A sentinel value must be a value that can never legitimately occur in the real data being summed (here, -1 is safe only because the numbers being totalled are assumed non-negative). If the data could genuinely include negative numbers, -1 would be a poor sentinel choice, since a real, valid input would be wrongly treated as the signal to stop.

7.4 Trace tables

A **trace table** is a table used to track how the value of every relevant variable changes, step by step, as an algorithm executes — without actually running the program on a computer. Tracing by hand is one of the most reliable ways to check that an algorithm's logic is correct, and to find exactly where a bug occurs when it is not.

Khái niệm cốt lõi

To construct a trace table: list one column for every variable that changes during execution (plus, where useful, a column for the current loop counter or condition being tested); then add one row for every pass through the loop (or every step of the algorithm), recording the new value of each variable *as soon as* it changes. The final row shows the state of every variable at the point the algorithm finishes, from which the output can be read off directly.

VN GIẢI THÍCH TIẾNG VIỆT

Bảng theo dõi biến (trace table) dùng để theo dõi giá trị của từng biến thay đổi như thế nào qua từng bước thực thi thuật toán, mà không cần chạy chương trình thật. Mỗi cột ứng với một biến; mỗi dòng ứng với một lần lặp hoặc một bước. Dòng cuối cùng cho biết trạng thái các biến khi thuật toán kết thúc, từ đó suy ra kết quả xuất ra.

7.4.1 Worked example: tracing a FOR loop

Ví dụ mẫu · Worked example

Trace the following pseudocode and state the final output.

```
Total <- 0
FOR Count <- 1 TO 5
   Total <- Total + Count
NEXT Count
OUTPUT Total
```

Count	Total
(before loop)	0
1	1
2	3
3	6
4	10
5	15

Trace of a FOR loop summing the integers 1 to 5.

The loop runs for `Count = 1, 2, 3, 4, 5` and then stops (since `Count` would next become 6, beyond the `TO 5` limit). The final value of `Total` is $1 + 2 + 3 + 4 + 5 = 15$, so the algorithm outputs **15**.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ở mỗi lần lặp, `Total` được cộng thêm giá trị hiện tại của `Count`. Bảng trace liệt kê giá trị của `Count` và `Total` sau mỗi lần lặp; dòng cuối cùng (khi `Count = 5`) cho kết quả xuất ra là **15**, đúng bằng tổng $1 + 2 + 3 + 4 + 5$.

7.4.2 Worked example: tracing a loop that searches an array

A trace table becomes more demanding once an algorithm processes an **array**, since each row must also show which array element is currently being examined.

Ví dụ mẫu · Worked example

The array `Scores` holds five values: `Scores` $= [42, 67, 15, 89, 53]$ (indices 1 to 5). Trace the pseudocode below and state the final output.

```
Max <- Scores[1]
FOR Index <- 2 TO 5
   IF Scores[Index] > Max THEN
      Max <- Scores[Index]
   ENDIF
NEXT Index
OUTPUT Max
```

Index	Scores[Index]	Scores[Index] > Max ?	Max
(before loop)	—	—	42
2	67	Yes	67
3	15	No	67
4	89	Yes	89
5	53	No	89

Trace of a FOR loop finding the maximum value in an array.

`Max` is initialised to the first element (42) before the loop begins, then the loop compares each remaining element in turn: 67 is larger than 42, so `Max` updates to 67; 15 is not larger than 67, so `Max` is unchanged; 89 is larger than 67, so `Max` updates to 89; 53 is not larger than 89, so `Max` stays at 89. The final output is **89**, correctly the largest value in the array.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khi thuật toán xử lý một **mảng (array)**, bảng trace cần thêm cột cho chỉ số phần tử đang xét (`Index`) và giá trị phần tử đó (`Scores[Index]`), cùng cột ghi lại kết quả kiểm tra điều kiện (`IF`). `Max` luôn được khởi tạo bằng phần tử **đầu tiên** của mảng — không phải bằng 0 — để thuật toán vẫn đúng ngay cả khi tất cả phần tử trong mảng là số âm.

7.5 Searching algorithms

Searching means locating a particular target value within a collection of data (an array or list). Two searching algorithms are required for Cambridge IGCSE Computer Science: **linear search** and **binary search**.

Khái niệm cốt lõi

Linear search checks every element of the list in turn, starting from the first, comparing it against the target value, until either the target is found or every element has been checked. It works correctly on data in *any* order (sorted or unsorted), but on a large list it can be slow, since in the worst case (target absent, or in the last position) it must examine every single element.

Binary search requires the data to already be **sorted**. It repeatedly compares the target to the *middle* element of the current search range: if they match, the target is found; if the target is smaller, the entire upper half of the range (including the middle element) can be discarded; if the target is larger, the entire lower half can be discarded. This halves the size of the remaining search range at every step, making binary search dramatically faster than linear search on large sorted data sets.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tìm kiếm tuần tự (linear search): kiểm tra lần lượt từng phần tử của danh sách từ đầu đến khi tìm thấy hoặc hết danh sách; hoạt động được với dữ liệu bất kỳ thứ tự nào (đã sắp xếp hay chưa), nhưng chậm với danh sách lớn. **Tìm kiếm nhị phân (binary search)**: yêu cầu dữ liệu **đã được sắp xếp**; liên tục so sánh với phần tử ở **giữa** phạm vi tìm kiếm hiện tại và loại bỏ một nửa phạm vi mỗi lần — nhanh hơn rất nhiều so với tìm kiếm tuần tự trên dữ liệu lớn.

7.5.1 Worked example: tracing a linear search

Ví dụ mẫu · Worked example

An unsorted list holds seven values: `List` $= [34, 12, 90, 8, 61, 23, 77]$ (indices 1 to 7). Trace a linear search for the target value 61.

```
Index <- 1
Found <- FALSE
WHILE Index <= 7 AND Found = FALSE
   IF List[Index] = Target THEN
      Found <- TRUE
      Position <- Index
   ELSE
      Index <- Index + 1
   ENDIF
ENDWHILE
```

Index	List[Index]	List[Index] = Target (61)?
1	34	No
2	12	No
3	90	No
4	8	No
5	61	**Yes — Found**

Linear search trace: target 61 found at position 5 after 5 comparisons.

The target is located at **position 5**, after exactly 5 comparisons. Because the list is unsorted,

linear search has no way to skip any element — every element up to and including the match must be checked individually.

7.5.2 Worked example: tracing binary search

Ví dụ mẫu · Worked example

The sorted array `List` $= [2, 7, 12, 18, 23, 31, 40, 55]$ (indices 1 to 8) is searched for the target value 23, using `Mid` $\leftarrow$ (`Low` + `High`) `DIV` 2.

Step	Low	High	Mid	List[Mid]
1	1	8	4	$18 - 23 > 18$, search upper half (`Low` $\leftarrow 5$)
2	5	8	6	$31 - 23 < 31$, search lower half (`High` $\leftarrow 5$)
3	5	5	5	$23 -$ **match found**

Binary search trace: target 23 found in 3 steps.

Only **3** comparisons are needed to search all 8 elements, because each step discards half of the remaining range.

7.5.3 Worked example: binary search on a larger array

Ví dụ mẫu · Worked example

The sorted array below has 16 elements (indices 1–16):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	8	14	19	25	31	38	44	50	57	63	70	76	83	89	95

Trace a binary search for the target value **95**.

Step	Low	High	Mid	Action
1	1	16	8	Array[8]$= 44$; $95 > 44 \Rightarrow$ `Low` $\leftarrow 9$
2	9	16	12	Array[12]$= 70$; $95 > 70 \Rightarrow$ `Low` $\leftarrow 13$
3	13	16	14	Array[14]$= 83$; $95 > 83 \Rightarrow$ `Low` $\leftarrow 15$
4	15	16	15	Array[15]$= 89$; $95 > 89 \Rightarrow$ `Low` $\leftarrow 16$
5	16	16	16	Array[16]$= 95 -$ **match found**

Binary search on a 16-element array: worst-case target requires 5 comparisons.

Because 95 is the *last* element, this is close to the algorithm's worst case for 16 elements; even so, only **5** comparisons were needed, versus up to 16 for a linear search on the same target.

Linear search has $O(n)$ worst-case performance — the number of comparisons needed grows in direct proportion to the number of items n. Binary search has $O(\log_2 n)$ worst-case performance — the number of comparisons only grows in proportion to how many times n can be halved. Doubling the size of the data roughly doubles the worst-case time for linear search, but only adds *one more* comparison to binary search's worst case.

VN **GIẢI THÍCH TIẾNG VIỆT**

Độ phức tạp $O(n)$ so với $O(\log_2 n)$: với tìm kiếm tuần tự, số phép so sánh trong trường hợp xấu nhất tỉ lệ thuận với số phần tử n (gọi là $O(n)$). Với tìm kiếm nhị phân, số phép so sánh trong trường hợp xấu nhất chỉ tỉ lệ với số lần n có thể được chia đôi (gọi là $O(\log_2 n)$). Khi dữ liệu tăng gấp đôi, tìm kiếm tuần tự cần gần gấp đôi số phép so sánh, trong khi tìm kiếm nhị phân chỉ cần thêm **đúng một** phép so sánh nữa — đây là lý do tìm kiếm nhị phân vượt trội hẳn trên dữ liệu lớn, miễn là dữ liệu đã được sắp xếp.

7.6 Sorting algorithms

Sorting rearranges the elements of a list into order (typically ascending). Two sorting algorithms are required: **bubble sort** and **insertion sort**.

Khái niệm cốt lõi

Bubble sort repeatedly passes through the list, comparing each pair of **adjacent** elements and swapping them if they are in the wrong order. Each full pass moves the largest remaining unsorted value one step closer to its correct position at the end of the list (it "bubbles up"). The algorithm stops once a complete pass makes **no swaps at all**, since this proves the list is now fully sorted.

Insertion sort builds up a sorted section of the list from left to right, one element at a time: it takes the next unsorted element and inserts it into its correct position *within* the already-sorted section, shifting larger elements one place to the right to make room.

VN **GIẢI THÍCH TIẾNG VIỆT**

Sắp xếp nổi bọt (bubble sort): liên tục duyệt qua danh sách, so sánh và hoán đổi từng cặp phần tử liền kề nếu sai thứ tự; mỗi lượt duyệt đẩy giá trị lớn nhất còn lại về cuối danh sách. Thuật toán dừng khi một lượt duyệt hoàn chỉnh không có hoán đổi nào — chứng tỏ danh sách đã được sắp xếp xong. **Sắp xếp chèn (insertion sort)**: xây dựng dần một phần đã sắp xếp từ trái sang phải, mỗi lần lấy phần tử tiếp theo và chèn nó vào đúng vị trí trong phần đã sắp xếp, dịch các phần tử lớn hơn sang phải để nhường chỗ.

7.6.1 Worked example: bubble sort (two passes)

Ví dụ mẫu · Worked example

Trace two passes of bubble sort on `List` $= [6, 2, 9, 4]$, comparing adjacent pairs left to right in each pass.

Pass 1: compare (6,2) — swap $\rightarrow [2, 6, 9, 4]$; compare (6,9) — no swap; compare (9,4) — swap $\rightarrow [2, 6, 4, 9]$. End of pass 1: $[2, 6, 4, 9]$ (2 swaps).

Pass 2: compare (2,6) — no swap; compare (6,4) — swap $\rightarrow [2, 4, 6, 9]$; compare (6,9) — no swap. End of pass 2: $[2, 4, 6, 9]$ (1 swap).

The list is now fully sorted: $[2, 4, 6, 9]$.

7.6.2 Worked example: bubble sort traced to completion

The example above stopped once the list happened to become sorted, but a full bubble sort (using a `Swapped` flag) always continues until an entire pass produces *no* swaps, which is how the algorithm itself recognises that sorting is complete.

Ví dụ mẫu · Worked example

Trace the following pseudocode, fully, on `List` $= [5, 1, 4, 2, 8]$ ($N = 5$), counting every comparison and swap.

```
REPEAT
   Swapped <- FALSE
   FOR i <- 1 TO N-1
      IF List[i] > List[i+1] THEN
         Temp <- List[i]
         List[i] <- List[i+1]
         List[i+1] <- Temp
         Swapped <- TRUE
      ENDIF
   NEXT i
UNTIL Swapped = FALSE
```

Pass	Comparisons made (i=1..4)	Swaps this pass	List after pass	Swapped
1	(5,1)swap,(5,4)swap,(5,2)swap,(5,8)no	3	$[1, 4, 2, 5, 8]$	TRUE
2	(1,4)no,(4,2)swap,(4,5)no,(5,8)no	1	$[1, 2, 4, 5, 8]$	TRUE
3	(1,2)no,(2,4)no,(4,5)no,(5,8)no	0	$[1, 2, 4, 5, 8]$	FALSE — stop

Full bubble sort trace: 3 passes, 12 total comparisons, 4 total swaps.

Working through Pass 1 in detail: List[1]=5, List[2]=1: $5 > 1$, swap → $[1, 5, 4, 2, 8]$. List[2]=5, List[3]=4: $5 > 4$, swap → $[1, 4, 5, 2, 8]$. List[3]=5, List[4]=2: $5 > 2$, swap → $[1, 4, 2, 5, 8]$. List[4]=5, List[5]=8: $5 > 8$ is false, no swap. Pass 1 ends at $[1, 4, 2, 5, 8]$ with 3 swaps, so `Swapped = TRUE` and the algorithm repeats.

Each of the 3 passes makes exactly 4 comparisons ($i = 1$ to $N - 1 = 4$), giving $4 \times 3 = 12$ comparisons in total. Swaps total $3 + 1 + 0 = 4$. The third pass is the **confirming pass**: it makes its 4 comparisons but finds nothing out of order, so `Swapped` stays `FALSE` and the algorithm correctly halts with the fully sorted list $[1, 2, 4, 5, 8]$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Phiên bản bubble sort dùng cờ `Swapped` sẽ lặp lại toàn bộ lượt duyệt cho đến khi có một lượt **không hoán đổi lần nào** — đây chính là "lượt xác nhận" (confirming pass) chứng minh danh sách đã sắp xếp xong. Ví dụ trên cần đúng 3 lượt, tổng cộng 12 phép so sánh và 4 lần hoán đổi.

7.6.3 Worked example: insertion sort

Ví dụ mẫu · Worked example

Trace insertion sort on `A` $= [9, 4, 7, 2]$ (indices 1–4).

Step (i)	Key inserted	Array after this step
$i = 2$	`key` $= 4$: shift 9 right, insert 4 at position 1	$[4, 9, 7, 2]$
$i = 3$	`key` $= 7$: shift 9 right, insert 7 at position 2	$[4, 7, 9, 2]$
$i = 4$	`key` $= 2$: shift 9, 7, 4 right, insert 2 at position 1	$[2, 4, 7, 9]$

Insertion sort trace: the sorted section (underlined conceptually) grows one element at a time.

At $i = 4$, `key`=2 is compared against 9 (shift right), then 7 (shift right), then 4 (shift right), then there is no element left to compare against, so 2 is inserted at position 1. The final sorted array is $[2, 4, 7, 9]$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Sắp xếp chèn xây dựng phần đã sắp xếp từ trái sang: ở bước $i = 4$, phần tử `key`=2 được so sánh lần lượt với các phần tử lớn hơn nó về bên trái (9, 7, 4), mỗi lần dịch phần tử đó sang phải một ô, cho đến khi tìm đúng vị trí chèn (đầu mảng). Đây là lý do insertion sort thường thực hiện **ít phép so sánh hơn** bubble sort khi dữ liệu gần như đã được sắp xếp sẵn — chỉ những phần tử thực sự sai vị trí mới cần dịch chuyển.

7.7 Validation and verification

Once an algorithm accepts data as input, it must also guard against incorrect or unusable data being entered. Cambridge IGCSE distinguishes two separate ideas that are often confused.

Khái niệm cốt lõi

Validation is an **automated check**, performed by the program itself, that input data is **reasonable or acceptable** according to predefined rules — for example, that a percentage lies between 0 and 100. Crucially, data can pass every validation check and still be factually **incorrect** (e.g. a student's real mark was 62%, but 26% was typed by mistake — 26 is a perfectly valid percentage, so validation would accept it without complaint).

Verification checks that data has been **entered or copied accurately**, i.e. that what ended up in the computer genuinely matches the original source. Two common verification methods are **double data entry** (the same data is typed independently twice, usually by two different people or on two occasions, and the computer compares the two copies for any mismatch) and **visual checking / proofreading** (a person compares the data on screen against the original source document by eye before it is accepted).

VN **GIẢI THÍCH TIẾNG VIỆT**

Kiểm tra hợp lệ (validation): một phép kiểm tra **tự động**, do chương trình thực hiện, để xem dữ liệu nhập vào có **hợp lý/chấp nhận được** hay không theo quy tắc đã định trước — nhưng dữ liệu có thể vượt qua mọi kiểm tra hợp lệ mà vẫn **sai về mặt thực tế**. **Xác minh (verification)**: kiểm tra xem dữ liệu đã được **nhập hoặc sao chép chính xác** hay chưa, tức là dữ liệu trong máy tính có đúng khớp với nguồn gốc ban đầu không — ví dụ **nhập dữ liệu hai lần (double entry)** hoặc **đối chiếu bằng mắt (visual check)** với tài liệu gốc.

(!) Lỗi thường gặp: Validation can never guarantee that data is *correct*, only that it is *reasonable*. A wrongly-typed mark of 26% instead of the true 62% will pass a range check (0–100) without being flagged, because 26 genuinely is a valid percentage — catching this kind of error is the job of *verification*, not validation.

7.7.1 Types of validation check

Check type	What it confirms
Range check	The value falls within a specified acceptable minimum and maximum.
Length check	The input contains an acceptable number of characters (a minimum, a maximum, or an exact count).
Type check	The input is of the expected data type (e.g. a whole number, not text).
Presence check	A required field has not been left empty.
Format check	The data matches a required pattern or fixed structure (e.g. 2 letters followed by 4 digits).
Check digit	An extra digit, calculated from the other digits using an agreed formula, is recalculated and compared to catch a mistyped or mistransmitted code.

Summary of validation check types.

Ví dụ mẫu · Worked example

Write pseudocode for a **range check** that a percentage mark, once entered, lies between 0 and 100 inclusive.

```
INPUT Mark
IF Mark >= 0 AND Mark <= 100 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: mark must be between 0 and 100"
ENDIF
```

Ví dụ mẫu · Worked example

Write pseudocode for a **length check** that a new password is between 8 and 12 characters long.

```
INPUT Password
IF LENGTH(Password) >= 8 AND LENGTH(Password) <= 12 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: password must be 8-12 characters"
ENDIF
```

Ví dụ mẫu · Worked example

Write pseudocode for a **type check** confirming that an entered age is a whole number.

```
INPUT Age
IF Age = INT(Age) THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: age must be a whole number"
ENDIF
```

`INT(Age)` discards any decimal part of `Age`; if the result equals `Age` itself, no decimal part was lost, so `Age` was already a whole number.

Ví dụ mẫu · Worked example

Write pseudocode for a **presence check** confirming that a required **Surname** field has not been left blank.

```
INPUT Surname
IF Surname = "" THEN
   OUTPUT "Error: surname cannot be left blank"
ELSE
   OUTPUT "Accepted"
ENDIF
```

Ví dụ mẫu · Worked example

Write pseudocode for a **format check** confirming that a reference code follows the required structure: exactly 2 letters followed by exactly 4 digits (e.g. `AB1234`).

```
INPUT RefCode
IF LENGTH(RefCode) = 6 AND ISALPHA(RefCode[1]) AND ISALPHA(RefCode[2])
   AND ISNUMERIC(RefCode[3]) AND ISNUMERIC(RefCode[4])
   AND ISNUMERIC(RefCode[5]) AND ISNUMERIC(RefCode[6]) THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: reference code must be 2 letters followed by 4 digits"
ENDIF
```

A format check does not confirm that a code *exists* or is genuine — only that it is arranged in the required pattern of letters and digits.

Ví dụ mẫu · Worked example

Recall the modulo-11 **check digit** method from Chapter 2: each of the leading digits is multiplied by a descending weight, the products are summed, and the check digit is derived from the remainder when this sum is divided by 11. Verify the check digit for the 4-digit code 3726 (digits 3, 7, 2, 6, using weights 5, 4, 3, 2 respectively).
Weighted sum $= (3\times5) + (7\times4) + (2\times3) + (6\times2) = 15 + 28 + 6 + 12 = 61$.
Remainder $= 61 \bmod 11$: since $11 \times 5 = 55$, remainder $= 61 - 55 = 6$.
Check digit $= 11 - 6 = 5$.
Verification: including the check digit with weight 1, the full weighted total is $61 + (5\times1) = 66$, and $66 \div 11 = 6$ exactly (remainder 0) — confirming the check digit **5** is correct. At the point of data entry, the check digit is **recalculated** from the digits as typed (or scanned) and compared to the check digit supplied; any mismatch means the code was captured inaccurately and should be rejected.

Ví dụ mẫu · Worked example

Write pseudocode that combines a **presence check** and a **range check** to validate an age input, which must not be left blank and must lie between 1 and 120 inclusive.

```
INPUT Age
IF Age = "" THEN
```

```
   OUTPUT "Error: age cannot be blank"
ELSE
   IF Age >= 1 AND Age <= 120 THEN
      OUTPUT "Accepted"
   ELSE
      OUTPUT "Error: age must be between 1 and 120"
   ENDIF
ENDIF
```

The presence check is placed *first*: if `Age` is blank, the range comparison is never even attempted, which avoids comparing an empty value against 1 and 120.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng trên tóm tắt sáu loại kiểm tra hợp lệ thường gặp: **range check** (giá trị nằm trong khoảng cho phép), **length check** (số ký tự hợp lệ), **type check** (đúng kiểu dữ liệu), **presence check** (trường bắt buộc không được để trống), **format check** (đúng cấu trúc/khuôn mẫu quy định), và **check digit** (chữ số kiểm tra tính từ các chữ số khác, dùng để phát hiện lỗi nhập/truyền dữ liệu — xem lại Chương 2). Ví dụ kết hợp cho thấy **presence check** nên được đặt trước để tránh so sánh một giá trị rỗng với khoảng số.

7.8 Testing algorithms

Writing validation rules is not enough on its own — the validation *code itself* must be tested to confirm it behaves correctly at every boundary and in every scenario. This requires deliberately designed **test data** in three categories.

Khái niệm cốt lõi

Normal data — typical, valid values that a user is expected to enter in ordinary use; these should be **accepted**.

Boundary data — values sitting **exactly at the edge** of an acceptable range (the smallest and largest values that should still be accepted); these test that the algorithm has not made an **off-by-one** error (e.g. using > instead of >=) and should still be **accepted**.

Erroneous data — invalid data that lies outside the acceptable range, is the wrong type, or is otherwise unacceptable; this data should be **rejected** by validation.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cần kiểm thử thuật toán bằng ba loại dữ liệu: **dữ liệu bình thường (normal)** — giá trị hợp lệ, điển hình, phải được **chấp nhận**; **dữ liệu biên (boundary)** — giá trị nằm **ngay tại ranh giới** của khoảng cho phép, dùng để kiểm tra lỗi **lệch một đơn vị (off-by-one)**, và vẫn phải được **chấp nhận**; **dữ liệu sai (erroneous)** — dữ liệu không hợp lệ (sai kiểu, ngoài khoảng, v.v.), phải bị **từ chối**.

Ví dụ mẫu · Worked example

Design a test data table for validating an input that must be a whole-number age from 1 to 120 inclusive.

Test data	Category	Expected outcome
45	Normal	Accepted (a typical, mid-range age)
1	Boundary (lower)	Accepted (the smallest valid age)
120	Boundary (upper)	Accepted (the largest valid age)
0	Erroneous	Rejected (one below the lower limit)
121	Erroneous	Rejected (one above the upper limit)
-5	Erroneous	Rejected (negative, far outside the range)
`"abc"`	Erroneous	Rejected (wrong data type, fails the type check)
`""` (blank)	Erroneous	Rejected (fails the presence check)

Test data categories for validating an age from 1 to 120.

The two boundary values (1 and 120) are the most important rows in the table: if the range check were mistakenly coded as `Age > 1 AND Age < 120`, both boundary tests would incorrectly fail, immediately exposing the off-by-one bug that normal-data testing alone would never reveal.

Boundary data is deliberately chosen to sit exactly *on* the edge of what should be accepted — *not* just outside it. Testing 1 and 120 (accepted) alongside 0 and 121 (rejected) together confirm that the inequality signs in the range check (>= and <=, rather than > and <) are exactly correct.

7.9 Putting the tools together: a complete worked case study

Every tool introduced in this chapter — decomposition, the IPO model, pseudocode, trace tables, and test data design — is normally applied *together*, in sequence, to a single sub-problem. This section works through one complete example from start to finish: the **"calculate overdue fines"** sub-task identified in the library loan system at the start of the chapter.

Khái niệm cốt lõi

A complete algorithm design, from a decomposed sub-task to a tested solution, typically follows these stages:

1. **Confirm the sub-task's scope** (what decomposition has isolated it as responsible for).
2. **Apply the IPO model** to state precisely what data comes in, what processing must happen, and what result must come out.
3. **Write pseudocode or a flowchart** implementing that processing, including any necessary validation of the input data.
4. **Trace the algorithm by hand** on at least one set of values to confirm the logic produces the correct result.
5. **Design test data** — normal, boundary and erroneous — to confirm the finished algorithm (and its validation) behaves correctly in every case, not just the one traced by hand.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một thiết kế thuật toán hoàn chỉnh thường trải qua năm bước: (1) xác định rõ phạm vi của bài toán con (từ bước phân rã); (2) áp dụng mô hình **IPO** để xác định chính xác đầu vào, xử lý và đầu ra; (3) viết mã giả/lưu đồ, bao gồm cả kiểm tra hợp lệ dữ liệu đầu vào cần thiết; (4) **trace** thuật toán bằng tay với ít nhất một bộ dữ liệu để xác nhận logic đúng; (5) thiết kế **dữ liệu kiểm**

thử (bình thường, biên, sai) để xác nhận thuật toán và phần kiểm tra hợp lệ hoạt động đúng trong mọi trường hợp, không chỉ trường hợp đã trace bằng tay.

Ví dụ mẫu · Worked example

Stage 1 — scope: the "calculate overdue fines" sub-task takes a single loan record that is known to be overdue and works out how much the borrower owes; it is not responsible for deciding *which* loans are overdue (that belongs to a different sub-task) or for accepting payment.

Stage 2 — IPO:

Stage	Detail
Input	Number of days a book is overdue (`DaysLate`); the fine rate charged per day (`DailyFineRate`).
Process	If `DaysLate` is greater than 0, multiply it by `DailyFineRate` to get the fine; otherwise the fine is 0.
Output	The fine amount owed by the borrower.

IPO breakdown of the overdue-fine calculation sub-task.

Stage 3 — pseudocode (with validation):

```
INPUT DaysLate
INPUT DailyFineRate
IF DaysLate < 0 THEN
   OUTPUT "Error: days late cannot be negative"
ELSE
   IF DaysLate > 0 THEN
      Fine <- DaysLate * DailyFineRate
   ELSE
      Fine <- 0
   ENDIF
   OUTPUT Fine
ENDIF
```

The range check (`DaysLate < 0`) is a validation step: a negative number of days late is never reasonable, since it would mean the book was returned before it was even due.

Stage 4 — trace for `DaysLate = 5` and `DailyFineRate = 0.20`:

Step	Value
`DaysLate`	5
Is `DaysLate` < 0?	No
Is `DaysLate` > 0?	Yes
`Fine <- DaysLate * DailyFineRate`	$5 \times 0.20 = 1.00$
Output	**1.00**

Trace of the overdue-fine algorithm for a typical late return.

Stage 5 — test data: `DaysLate = 5` (normal, fine $= 1.00$); `DaysLate = 0` (boundary, fine $= 0$, exactly on time); `DaysLate = 1` (boundary, the smallest genuinely late value, fine $= 0.20$); `DaysLate = -3` (erroneous, correctly rejected by the range check).

(!) Lỗi thường gặp: It is tempting to treat trace tables and test data design as the *same* activity, but they serve different purposes. A trace table confirms the *logic* is correct for one specific, hand-chosen set of values during design. Test data design is broader and happens *after* the algorithm is written: it deliberately includes boundary and erroneous values — which a designer tracing "typical" values by hand might never think to check — specifically to catch validation bugs (such as an off-by-one error in a range check) that a single normal-data trace would never reveal.

7.10 Practice

Bài tập luyện tập

Trace the pseudocode below and determine the final value output.

```
Result <- 1
FOR Count <- 1 TO 4
   Result <- Result * Count
NEXT Count
OUTPUT Result
```

(A) 4
(B) 10
(C) 24
(D) 120

Lời giải chi tiết

Tracing: Count=1, Result=$1\times1 = 1$; Count=2, Result=$1\times2 = 2$; Count=3, Result=$2\times3 = 6$; Count=4, Result=$6\times4 = 24$. The loop ends after Count=4. Output is **24** (which is $4! = 4\times3\times2\times1$). **(C)**.

Bài tập luyện tập

Decomposition and abstraction are both used when designing an algorithm for a large problem. Which statement correctly distinguishes them?

(A) Decomposition ignores irrelevant detail; abstraction splits the problem into sub-problems

(B) Decomposition splits a large problem into smaller sub-problems; abstraction decides which details of the problem are relevant and which can be ignored

(C) They are two names for exactly the same process

(D) Decomposition is only used for flowcharts; abstraction is only used for pseudocode

Lời giải chi tiết

Decomposition breaks a large, complex problem down into smaller, more manageable sub-problems. **Abstraction** is a separate skill: within any given problem or sub-problem, it decides which details actually matter to the solution and which real-world details can safely be ignored. **(B)**.

Bài tập luyện tập

An "online pizza ordering system" is too large a problem to design a single algorithm for directly. (a) Decompose it into at least four independent sub-tasks. (b) For one of your sub-tasks, give one example of abstraction — a real-world detail the algorithm can safely ignore.

Lời giải chi tiết

(a) Possible decomposition into sub-tasks: **browse the menu and select items**; **customise an order** (size, toppings); **calculate the order total**, including any delivery charge; **take payment details**; **confirm the order and estimate a delivery time**. Each of these can be designed, coded and tested as an independent algorithm.

(b) For **"calculate the order total"**, the algorithm needs the price of each selected item and any delivery charge, but can completely ignore irrelevant real-world details such as the customer's favourite topping in previous orders, the colour of the delivery driver's car, or the weather outside — none of these affect the total price calculation.

Bài tập luyện tập

Apply the IPO model to a program that converts a temperature from degrees Celsius to degrees Fahrenheit, using $F = \frac{9}{5}C + 32$. State one input, one process step, and one output.

Lời giải chi tiết

Input: the temperature in degrees Celsius. **Process:** multiply the Celsius value by $\frac{9}{5}$, then add 32 to the result. **Output:** the calculated temperature in degrees Fahrenheit.

Bài tập luyện tập

In a flowchart, which shape is used to represent a decision, where the flow of control can branch along more than one path?

(A) Rectangle
(B) Oval
(C) Diamond
(D) Parallelogram

Lời giải chi tiết

A **diamond** represents a decision — a condition that is tested, with the flow of control branching (typically "Yes"/"No") depending on the result. Rectangles represent process steps; ovals mark start/stop; parallelograms represent input/output. **(C)**.

Bài tập luyện tập

Trace the following nested-loop pseudocode fully and state the value output.

```
Total <- 0
FOR Row <- 1 TO 3
   FOR Col <- 1 TO 2
      Total <- Total + (Row * Col)
   NEXT Col
NEXT Row
OUTPUT Total
```

Lời giải chi tiết

Row	Col	Row × Col	Total
(before loop)	—	—	0
1	1	1	1
1	2	2	3
2	1	2	5
2	2	4	9
3	1	3	12
3	2	6	18

Trace of a nested FOR loop.

For every `Row` (1 to 3), the inner loop runs `Col` from 1 to 2, adding `Row` × `Col` to `Total` each time. The inner loop therefore executes $3 \times 2 = 6$ times in total. The final output is **18**.

Bài tập luyện tập

The array `Values` $= [12, 45, 7, 45, 30]$ (indices 1–5) is processed by the "find the maximum" pseudocode from the theory section (`Max <- Values[1]`, then compare `Values[Index] > Max` for `Index <- 2 TO 5`). Trace it fully and state the output.

Lời giải chi tiết

Index	Values[Index]	Values[Index] > Max ?	Max
(before loop)	—	—	12
2	45	Yes	45
3	7	No	45
4	45	No ($45 > 45$ is false)	45
5	30	No	45

Trace of the maximum-finding algorithm with a repeated value.

The output is **45**. Note the row for Index=4: the condition uses a *strict* greater-than test, so a value *equal to* the current `Max` does not trigger an update — `Max` correctly keeps the value found earlier, and the repeated 45 does not cause any error.

Bài tập luyện tập

A binary search is used on a sorted array of 1000 elements. What is the approximate maximum number of comparisons needed in the worst case?

(A) 10
(C) 500
(B) 100
(D) 1000

Lời giải chi tiết

Binary search halves the search range at each step, so the worst-case number of comparisons is approximately $\log_2 n$. Since $2^9 = 512$ and $2^{10} = 1024$, $\log_2 1000$ is just under 10, so at most about **10** comparisons are ever needed — versus up to 1000 for a linear search on the same data. **(A)**.

Bài tập luyện tập

A sorted array of 12 elements is $[4, 9, 15, 22, 28, 35, 41, 48, 54, 60, 67, 73]$ (indices 1–12). Trace a binary search for the target value 30, which is **not present** in the array, and state the final outcome.

Lời giải chi tiết

Step	Low	High	Mid	Action
1	1	12	6	Array[6]$= 35$; $30 < 35 \Rightarrow$ `High` $\leftarrow 5$
2	1	5	3	Array[3]$= 15$; $30 > 15 \Rightarrow$ `Low` $\leftarrow 4$
3	4	5	4	Array[4]$= 22$; $30 > 22 \Rightarrow$ `Low` $\leftarrow 5$
4	5	5	5	Array[5]$= 28$; $30 > 28 \Rightarrow$ `Low` $\leftarrow 6$
5	6	5	—	`Low` $>$ `High`: search range is empty

Binary search trace for an absent target: the algorithm correctly reports "not found".

Once `Low` (6) exceeds `High` (5), there is no remaining range left to search, so the algorithm terminates and reports the target **not found**, after 4 comparisons.

Bài tập luyện tập

A school stores 60 student names in the order they enrolled (not alphabetically). Which search approach should be used to find a particular name, and why?

(A) Binary search, because it is always the faster algorithm

(B) Linear search, because the data is not sorted and binary search requires sorted data

(C) Either algorithm gives identical performance on any data

(D) Neither algorithm can search text data such as names

Lời giải chi tiết

Binary search only works correctly on **sorted** data, since it relies on being able to discard half of the remaining range based on a single comparison. Since the 60 names are in enrolment order, not alphabetical order, binary search cannot be used directly (the list would first have to be sorted, at extra cost). **Linear search** works correctly on data in any order, making it the appropriate choice here despite being slower on large sorted data. **(B)**.

Bài tập luyện tập

A linear search is performed on `List` $= [5, 17, 9, 23, 41]$ (indices 1–5) for the target value 30, which is not present. Trace every comparison made and state the final outcome.

Lời giải chi tiết

Index	List[Index]	List[Index] $= 30$?
1	5	No
2	17	No
3	9	No
4	23	No
5	41	No

Linear search trace for an absent target: all 5 elements are checked.

Every element is compared (since none match), so the algorithm reaches the end of the list having made **5** comparisons, and correctly reports the target **not found**. Unlike binary search, linear search has no way to know the target is absent until it has examined every single element.

Bài tập luyện tập

The following pseudocode is intended to find the maximum value in the array `List` (which may contain negative numbers), for `Index` from 1 to `N`.

```
Max <- 0
FOR Index <- 1 TO N
   IF List[Index] > Max THEN
      Max <- List[Index]
   ENDIF
NEXT Index
OUTPUT Max
```

Identify the bug, explain a set of data that reveals it, and correct the pseudocode.

Lời giải chi tiết

Bug: `Max` is initialised to 0. If every value in `List` is negative (e.g. $[-8, -3, -15, -1]$), no element will ever be greater than 0, so the `IF` condition is never true and `Max` incorrectly remains 0 — a value that is not even present in the list, and is larger than every genuine element.

Fix: initialise `Max` to the *first element of the list* rather than 0, and begin the loop from the second element (since the first has already been used to initialise `Max`):

```
Max <- List[1]
FOR Index <- 2 TO N
   IF List[Index] > Max THEN
      Max <- List[Index]
   ENDIF
NEXT Index
OUTPUT Max
```

This version works correctly regardless of whether the list contains positive numbers, negative numbers, or a mixture of both.

Bài tập luyện tập

(a) A bubble sort is applied to a list of 6 elements. How many comparisons are made during the *first* pass? (b) Explain why insertion sort may perform noticeably *fewer* comparisons than bubble sort when the data is already nearly sorted.

Lời giải chi tiết

(a) A single pass of bubble sort compares each pair of adjacent elements once. For a list of $n = 6$ elements, there are $n - 1 = 5$ adjacent pairs, so the first pass makes **5** comparisons.

(b) Insertion sort only shifts an element as far as it needs to go to reach its correct position

among the elements already placed — if the data is nearly sorted, most new elements are already close to (or exactly at) their correct position, so very few comparisons and shifts are needed per element (best case approaches $O(n)$). A simple bubble sort (without the `Swapped` flag early exit) instead always performs a full pass of $n - 1$ comparisons regardless of how sorted the data already is, so it does not automatically benefit from data being nearly sorted in the same way.

Bài tập luyện tập

Trace insertion sort fully on `A` $= [15, 6, 12, 3, 9]$ (indices 1–5), showing the array after each element is inserted.

Lời giải chi tiết

Step (i)	Key inserted	Array after this step
$i = 2$	`key` $= 6$: shift 15 right, insert 6 at position 1	$[6, 15, 12, 3, 9]$
$i = 3$	`key` $= 12$: shift 15 right, insert 12 at position 2	$[6, 12, 15, 3, 9]$
$i = 4$	`key` $= 3$: shift 15, 12, 6 right, insert 3 at position 1	$[3, 6, 12, 15, 9]$
$i = 5$	`key` $= 9$: shift 15, 12 right (6 is not > 9), insert 9 at position 3	$[3, 6, 9, 12, 15]$

Full insertion sort trace on a 5-element array.

At $i = 5$, `key`=9 is compared to 15 (shift right), then 12 (shift right), then 6 (not greater than 9, stop shifting), so 9 is inserted immediately after 6, at position 3. The final sorted array is $[3, 6, 9, 12, 15]$.

Bài tập luyện tập

A search engine must repeatedly search a fixed, alphabetically sorted list of one million words to check spelling. Which searching algorithm is more suitable, and why?

(A) Linear search, since it checks every element and is therefore more thorough

(B) Binary search, since the list is sorted and its $O(\log_2 n)$ performance makes searching a million-element list dramatically faster than linear search's $O(n)$ performance

(C) Bubble sort, since it is designed specifically for searching

(D) Neither algorithm can be used on text data

Lời giải chi tiết

The word list is already sorted, so binary search's requirement is satisfied. Its $O(\log_2 n)$ worst-case performance means a list of one million ($2^{20} \approx 10^6$) words needs only around 20 comparisons in the worst case, versus up to one million for linear search — a dramatic difference at this scale. **(B)**.

Bài tập luyện tập

Explain the difference between **validation** and **verification**, using the scenario of a clerk typing a customer's 11-digit phone number into a database.

Lời giải chi tiết

Validation would be an automated check built into the data-entry program — for example, a length check confirming exactly 11 digits were entered, and a type check confirming only digits (no letters) were entered. This check runs automatically and would accept any 11-digit numeric string, even if it is not the customer's *actual* phone number.

Verification confirms the number was **copied accurately** from its original source (e.g. a customer registration form). This could be done by **double data entry** — a second person types the same number independently, and the computer compares the two entries, flagging any mismatch — or by the clerk **visually proofreading** the typed number against the original form before saving. Verification is what would catch a single mistyped digit that validation's range/length/type checks would never notice, since a mistyped 11-digit number is still a perfectly valid 11-digit number.

Bài tập luyện tập

Write pseudocode for a **range check** confirming that a temperature reading from an automatic weather station sensor lies between -20 and 50 degrees Celsius inclusive.

Lời giải chi tiết

```
INPUT Temperature
IF Temperature >= -20 AND Temperature <= 50 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: temperature must be between -20 and 50"
ENDIF
```

The check uses >= and <= (not strict > and <) so that the boundary values -20 and 50 themselves are correctly accepted rather than rejected.

Bài tập luyện tập

Write pseudocode for a **length check** confirming that a new username is between 5 and 15 characters long.

Lời giải chi tiết

```
INPUT Username
IF LENGTH(Username) >= 5 AND LENGTH(Username) <= 15 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: username must be 5-15 characters"
ENDIF
```

Bài tập luyện tập

Write pseudocode for a **type check** confirming that the number of tickets a customer wishes to purchase is a whole number.

Lời giải chi tiết

```
INPUT Tickets
IF Tickets = INT(Tickets) THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: number of tickets must be a whole number"
ENDIF
```

If `Tickets` contains a fractional part (e.g. 2.5), `INT(Tickets)` truncates it to 2, which no longer equals the original 2.5, correctly failing the check.

Bài tập luyện tập

An online form has a required **"Email address"** field. A user submits the form having left this field completely empty. Which validation check would catch this, and what would it typically be combined with in a real system to confirm the email is properly structured?

(A) A range check alone

(B) A presence check to catch the blank field, typically combined with a format check to confirm the structure (e.g. text `@` text . text)

(C) A check digit alone

(D) A length check alone, since format is irrelevant for email addresses

Lời giải chi tiết

A **presence check** specifically confirms that a required field has not been left blank, which is exactly the fault here. In a real system this is normally combined with a **format check**, which confirms the entered text follows the general structure of an email address (some text, an `@` symbol, more text, a full stop, and a domain ending) — catching a different kind of error (a non-blank but badly-structured entry) that a presence check alone would miss. **(B)**.

Bài tập luyện tập

A student ID must follow the pattern "2 letters followed by 4 digits" (e.g. `JS4821`). Which type of validation check specifically confirms that entered data follows this required pattern?

(A) Range check

(B) Presence check

(C) Format check

(D) Check digit

Lời giải chi tiết

A **format check** confirms that data matches a required pattern or fixed structure — exactly the requirement here (2 letters, then 4 digits, in that specific order). A range check tests numeric boundaries; a presence check tests only whether a field is blank; a check digit is a separately calculated extra digit used to catch entry/transmission errors. **(C)**.

Bài tập luyện tập

A 4-digit code has digits 8, 1, 4, 9 with weights 5, 4, 3, 2 respectively, using the modulo-11 check-digit method. (a) Calculate the check digit. (b) The code is later mistyped with the first two digits transposed (giving 1, 8, 4, 9), but the printed check digit is not changed. Show

whether recalculating with the original check digit detects this error.

Lời giải chi tiết

(a) Weighted sum $= (8\times5) + (1\times4) + (4\times3) + (9\times2) = 40 + 4 + 12 + 18 = 74$.
Remainder $= 74 \bmod 11$: since $11 \times 6 = 66$, remainder $= 74 - 66 = 8$.
Check digit $= 11 - 8 =$ **3**. *Verify:* $74 + (3\times1) = 77 = 11 \times 7$, remainder 0 — confirmed correct.

(b) With the transposed digits (1, 8, 4, 9) and the same weights (5, 4, 3, 2): weighted sum $= (1\times5) + (8\times4) + (4\times3) + (9\times2) = 5 + 32 + 12 + 18 = 67$. Adding the (unchanged, originally printed) check digit with weight 1: $67 + (3\times1) = 70$. $70 \div 11 = 6$ remainder 4 (since $11 \times 6 = 66$). Because the remainder is not 0, the code is correctly flagged as **invalid** — the transposition has been detected. This works because the two swapped digits carry *different* weights (5 and 4), so swapping them changes the weighted sum; a check digit calculated with equal weights for every position would not reliably catch this kind of error.

Bài tập luyện tập

Using the combined presence-and-range-check pseudocode for age (must not be blank, must be between 1 and 120), state the output for each of these three inputs: (i) `""` (blank), (ii) `150`, (iii) `45`.

Lời giải chi tiết

(i) `Age = ""`: the presence check (`IF Age = ""`) is true first, so the program outputs **"Error: age cannot be blank"**, without ever reaching the range check.
(ii) `Age = 150`: not blank, so the range check runs; $150 \leq 120$ is false, so the program outputs **"Error: age must be between 1 and 120"**.
(iii) `Age = 45`: not blank, and $1 \leq 45 \leq 120$ is true, so the program outputs **"Accepted"**.

Bài tập luyện tập

Design a test data table (test data, category, expected outcome) for validating a percentage score that must be a whole number from 0 to 100 inclusive. Include at least one normal, two boundary, and three erroneous values.

Lời giải chi tiết

Test data	Category	Expected outcome
50	Normal	Accepted
0	Boundary (lower)	Accepted
100	Boundary (upper)	Accepted
-1	Erroneous	Rejected (below the lower limit)
101	Erroneous	Rejected (above the upper limit)
`"pass"`	Erroneous	Rejected (wrong data type)

Test data categories for validating a percentage score from 0 to 100.

The boundary values (0 and 100) are essential: they confirm the range check correctly uses `>= 0` and `<= 100` rather than strict inequalities that would wrongly reject a genuinely valid score of exactly 0 or 100.

Bài tập luyện tập

An age-validation check accepts whole numbers from 1 to 120 inclusive. A tester submits the value 120. Which category of test data does this represent?

(A) Normal data

(B) Boundary data

(C) Erroneous data

(D) None of the above — 120 should not be tested at all

Lời giải chi tiết

The value 120 sits exactly at the **upper edge** of the acceptable range (1 to 120), rather than being a typical, unremarkable value from the middle of the range. This makes it **boundary data** — used specifically to confirm the algorithm accepts the largest value it should, rather than incorrectly rejecting it due to an off-by-one error. **(B)**.

Bài tập luyện tập

A range check is intended to accept whole-number ages from 1 to 120 inclusive, but has been coded as follows:

```
IF Age > 1 AND Age < 120 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: age must be between 1 and 120"
ENDIF
```

Identify the bug using boundary test data, and correct the pseudocode.

Lời giải chi tiết

Bug: the comparisons use *strict* inequalities (> and <) instead of >= and <=. Testing the boundary values reveals this immediately: `Age = 1` gives $1 > 1$, which is **false**, so the valid boundary value 1 is wrongly rejected; likewise `Age = 120` gives $120 < 120$, also **false**, so the valid boundary value 120 is wrongly rejected too.

Fix:

```
IF Age >= 1 AND Age <= 120 THEN
   OUTPUT "Accepted"
ELSE
   OUTPUT "Error: age must be between 1 and 120"
ENDIF
```

With >= and <=, both boundary values 1 and 120 are now correctly accepted.

Bài tập luyện tập

The pseudocode below is intended to sum only the **even** numbers in `List` $= [2, 5, 8, 11, 4]$ (indices 1–5), but it produces the wrong output.

```
Total <- 0
FOR Index <- 1 TO 5
   IF List[Index] MOD 2 = 1 THEN
```

```
      Total <- Total + List[Index]
   ENDIF
NEXT Index
OUTPUT Total
```

Trace the pseudocode, state the output it actually produces, and explain the bug.

Lời giải chi tiết

Index	List[Index]	List[Index] MOD 2 = 1 ?	Total
(before loop)	–	–	0
1	2	No (2 MOD 2 = 0)	0
2	5	Yes (5 MOD 2 = 1)	5
3	8	No	5
4	11	Yes	16
5	4	No	16

Debugging trace: the algorithm sums odd numbers instead of even numbers.

The actual output is **16** ($5+11$), not the intended 14 ($2+8+4$). The trace shows exactly where the logic is wrong: `List[Index] MOD 2 = 1` is true only for **odd** numbers (remainder 1 when divided by 2), so the condition currently selects odd numbers rather than even ones. The fix is to test `List[Index] MOD 2 = 0` instead, which is true precisely when a number is exactly divisible by 2 (even).

Bài tập luyện tập

Which statement best describes the purpose of **abstraction** in algorithm design, as distinct from decomposition?

(A) Abstraction focuses only on details relevant to the problem, deliberately ignoring irrelevant real-world complexity

(B) Abstraction is the process of splitting a large problem into smaller sub-problems

(C) Abstraction means translating pseudocode into a programming language

(D) Abstraction means testing an algorithm with boundary data

Lời giải chi tiết

Abstraction focuses a solution only on the details that are actually relevant to solving the problem, deliberately setting aside real-world complexity that does not affect the outcome. (Splitting a problem into sub-problems is decomposition; translating pseudocode into code is implementation; testing with boundary data is part of testing, not abstraction.) **(A)**.

Bài tập luyện tập

Using the library loan system decomposition (add a book, issue a loan, return a book, search the catalogue, calculate overdue fines), write top-down pseudocode for a main menu that calls the correct sub-task based on the user's numbered choice.

Lời giải chi tiết

```
INPUT Choice
CASE OF Choice
   1: CALL AddBook
   2: CALL IssueLoan
   3: CALL ReturnBook
   4: CALL SearchCatalogue
   5: CALL CalculateOverdueFines
   OTHERWISE: OUTPUT "Invalid choice"
ENDCASE
```

This top-level pseudocode maps the decomposition/structure diagram directly onto structured pseudocode: each leaf of the diagram becomes one independent subroutine, called from a single, simple main routine. Each subroutine (`AddBook`, `IssueLoan`, etc.) can then be designed, written and tested completely separately, without needing to understand the internal logic of any of the others.

Programming

Mục tiêu Unit: Kiểu dữ liệu, biến và hằng số (data types, variables, constants); các phép toán số học, so sánh và logic (arithmetic, relational, logical operators); ba cấu trúc lập trình cơ bản — tuần tự, rẽ nhánh (IF, CASE), lặp (FOR, WHILE, REPEAT); xử lý chuỗi ký tự (LENGTH, SUBSTRING, nối chuỗi, UCASE/LCASE); mảng một chiều và hai chiều, bao gồm thuật toán tìm kiếm tuần tự; thủ tục (procedure) và hàm (function), truyền tham số theo giá trị (by value) và theo tham chiếu (by reference), phạm vi biến (variable scope); và thao tác với tệp văn bản (mở, đọc, ghi, đóng tệp).

8.1 Data types, variables and constants

Every program works by storing, changing and combining pieces of data in the computer's memory. Before any of that can happen, the programmer must say exactly what *kind* of data each piece of storage will hold, and give that storage a name so it can be referred to throughout the program.

Khái niệm cốt lõi

A **variable** is a named location in memory whose value can change while the program is running. In Cambridge pseudocode, a variable must be declared before use with the keyword `DECLARE`, in the form `DECLARE <identifier> : <data type>`. The six data types used throughout Cambridge 0478 pseudocode are:

- **INTEGER** — a whole number, positive, negative or zero (e.g. -3, 0, 42).
- **REAL** — a number that may include a fractional (decimal) part (e.g. 3.14, -0.5, 100.0).
- **CHAR** — a single character (e.g. `'A'`, `'7'`, `'$'`), always written in single quotes.
- **STRING** — a sequence of zero or more characters (e.g. `"Hello"`, `"IGCSE"`), always written in double quotes.
- **BOOLEAN** — one of exactly two values, `TRUE` or `FALSE`.
- **DATE** — a calendar date (e.g. `01/09/2026`).

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **biến (variable)** là một ô nhớ có tên, giá trị của nó có thể thay đổi trong quá trình chương trình chạy. Trong mã giả Cambridge, biến phải được khai báo trước khi dùng bằng từ khoá `DECLARE`, theo cú pháp `DECLARE <tên biến> : <kiểu dữ liệu>`. Sáu kiểu dữ liệu cơ bản là: **INTEGER** (số nguyên), **REAL** (số thực có phần thập phân), **CHAR** (một ký tự duy nhất, đặt trong dấu nháy đơn), **STRING** (chuỗi ký tự, đặt trong dấu nháy kép), **BOOLEAN** (chỉ nhận giá trị TRUE hoặc FALSE), và **DATE** (ngày tháng).

Ví dụ mẫu · Worked example

Declare appropriate variables for a program that records a student's exam mark (a whole number out of 100), their average percentage across the year (which may include decimals), their

initial (a single letter), whether they passed, and the date they sat the exam.

```
DECLARE ExamMark      : INTEGER
DECLARE AveragePercent : REAL
DECLARE Initial       : CHAR
DECLARE HasPassed     : BOOLEAN
DECLARE ExamDate      : DATE
```

`ExamMark` is `INTEGER` because a raw mark out of 100 is always a whole number; `AveragePercent` is `REAL` because an average (e.g. a division of several marks) will typically include a fractional part; `Initial` is `CHAR` because it holds exactly one letter; `HasPassed` is `BOOLEAN` because it can only ever be `TRUE` or `FALSE`; `ExamDate` is `DATE`.

(!) Lỗi thường gặp: Do not confuse `CHAR` and `STRING`. A `CHAR` can only ever hold exactly one character and is written with *single* quotes, e.g. `'A'`. A `STRING` holds a sequence of any number of characters (including zero) and is written with *double* quotes, e.g. `"A"` or `"Cambridge"`. Assigning a multi-character value such as `"AB"` to a `CHAR` variable is a type error.

8.1.1 Constants

Some values in a program never change while the program is running — a tax rate, the number of days in a week, the value of π used in an area calculation. These are best declared as constants rather than ordinary variables.

Khái niệm cốt lõi

A **constant** is a named value that is fixed at the start of the program and *cannot be changed* while the program executes. In Cambridge pseudocode a constant is declared with the keyword `CONSTANT`, in the form `CONSTANT <identifier> = <value>` (note: `=`, not `<-`, because a constant is never re-assigned). Constants are used instead of writing the raw value (a so-called **"magic number"**) directly in the code, for three main reasons:

- **Readability** — `CONSTANT Pi = 3.14159` followed by `Area <- Pi * Radius * Radius` is far clearer to a human reader than `Area <- 3.14159 * Radius * Radius`, where the meaning of the number is not obvious.
- **Single point of update** — if the value needs to change (e.g. a VAT rate rises from 20% to 22.5%), it only has to be edited in the one `CONSTANT` declaration, rather than searched for and changed everywhere it was typed directly into the code.
- **Error prevention** — because the value cannot be reassigned elsewhere in the program, a constant protects against an accidental line of code overwriting a value that was never meant to change.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **hằng số (constant)** là một giá trị được đặt tên và cố định ngay từ đầu chương trình, *không thể thay đổi* trong suốt quá trình chạy. Hằng số được khai báo bằng từ khoá `CONSTANT`, theo cú pháp `CONSTANT <tên> = <giá trị>` (dùng dấu = chứ không phải <-, vì hằng số không bao giờ được gán lại). Dùng hằng số thay vì viết trực tiếp một con số "ma thuật" (magic number) giúp: (1) mã nguồn dễ đọc hơn; (2) chỉ cần sửa một chỗ duy nhất nếu giá trị thay đổi; (3) tránh lỗi do vô tình gán lại một giá trị lẽ ra không bao giờ được thay đổi.

Ví dụ mẫu · Worked example

Write pseudocode that declares a constant for the value of π and uses it to calculate the area of a circle of radius 5.

```
CONSTANT Pi = 3.14159
DECLARE Radius : REAL
DECLARE Area   : REAL
Radius <- 5
Area <- Pi * Radius * Radius
OUTPUT Area
```

Tracing: `Radius` $= 5$, so `Area` $= 3.14159 \times 5 \times 5 = 3.14159 \times 25 = 78.53975$. The program outputs **78.53975**.

A `CONSTANT` is fixed for the entire run of the program and is set once, in its declaration. A `VARIABLE` declared with `DECLARE` may be assigned and reassigned as many times as the program logic requires using the <– assignment operator. Attempting to assign a new value to a constant (e.g. `Pi <- 3.2`) is not permitted.

8.2 Operators

Once data is stored in variables, a program needs ways to combine, compare and calculate with it. Cambridge pseudocode groups these into three families: arithmetic operators (for calculations), relational operators (for comparisons) and logical operators (for combining true/false conditions).

8.2.1 Arithmetic operators

Khái niệm cốt lõi

Arithmetic operators perform calculations on numeric (`INTEGER` or `REAL`) values.

Operator	Meaning	Example
+	Addition	$5 + 3 = 8$
–	Subtraction	$5 - 3 = 2$
*	Multiplication	$5 * 3 = 15$
/	Real (decimal) division	$5/2 = 2.5$
DIV	Integer division — the whole-number quotient, remainder discarded	5 `DIV` $2 = 2$
MOD	Modulus — the remainder left over after integer division	5 `MOD` $2 = 1$

Arithmetic operators in Cambridge pseudocode.

`DIV` and `MOD` always work together: for any two positive integers a and b, $a = (a \texttt{ DIV } b) \times b + (a \texttt{ MOD } b)$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Các phép toán số học: + (cộng), – (trừ), * (nhân), / (chia lấy kết quả thập phân), `DIV` (chia lấy phần nguyên — thương số, bỏ phần dư), `MOD` (chia lấy phần dư — số dư còn lại sau phép chia nguyên). Hai phép `DIV` và `MOD` luôn đi cùng nhau: với hai số nguyên dương a và b bất kỳ, ta luôn có $a = (a \texttt{ DIV } b) \times b + (a \texttt{ MOD } b)$.

Ví dụ mẫu · Worked example

Evaluate 23 DIV 5 and 23 MOD 5, then evaluate 17 DIV 4 and 17 MOD 4.

```
DECLARE A, B : INTEGER
A <- 23 DIV 5
B <- 23 MOD 5
OUTPUT A
OUTPUT B
```

$23/5 = 4.6$, so the whole-number quotient is 4: `23 DIV 5` $= 4$. The remainder is $23 - (4 \times 5) = 23 - 20 = 3$: `23 MOD 5` $= 3$. Check: $4 \times 5 + 3 = 23$. □

Similarly, $17/4 = 4.25$, so `17 DIV 4` $= 4$, and the remainder is $17 - (4 \times 4) = 17 - 16 = 1$, so `17 MOD 4` $= 1$. `DIV` and `MOD` are frequently used together — for example, `MOD 2` tests whether a number is even (remainder 0) or odd (remainder 1), and `DIV 60` converts a number of seconds into whole minutes.

(!) Lỗi thường gặp: `/` and `DIV` are not interchangeable. `7 / 2` evaluates to the `REAL` value 3.5, but `7 DIV 2` evaluates to the `INTEGER` value 3 (the remainder 1 is simply discarded, *not* rounded). Using `/` where whole-number output is required (e.g. splitting minutes into hours and minutes) will produce a decimal value that then needs extra code to fix, whereas `DIV` gives the correct whole number directly.

8.2.2 Relational (comparison) operators

Khái niệm cốt lõi

Relational operators compare two values and produce a `BOOLEAN` result (`TRUE` or `FALSE`). They are used almost exclusively inside the conditions of `IF`, `WHILE` and `REPEAT...UNTIL` statements.

Operator	Meaning	Example (evaluates to)
=	Equal to	$5 = 5$ is `TRUE`
<>	Not equal to	$5 <> 5$ is `FALSE`
<	Less than	$3 < 5$ is `TRUE`
>	Greater than	$3 > 5$ is `FALSE`
<=	Less than or equal to	$5 <= 5$ is `TRUE`
>=	Greater than or equal to	$4 >= 5$ is `FALSE`

Relational operators in Cambridge pseudocode.

VN **GIẢI THÍCH TIẾNG VIỆT**

Các phép so sánh (relational operators): = (bằng), <> (khác), < (nhỏ hơn), > (lớn hơn), <= (nhỏ hơn hoặc bằng), >= (lớn hơn hoặc bằng). Kết quả của một phép so sánh luôn là một giá trị `BOOLEAN`: `TRUE` hoặc `FALSE`. Các phép so sánh gần như luôn xuất hiện bên trong điều kiện của `IF`, `WHILE`, hoặc `REPEAT...UNTIL`.

8.2.3 Logical operators

Khái niệm cốt lõi

Logical operators — `AND`, `OR` and `NOT` — combine one or more `BOOLEAN` values (often the results of relational comparisons) into a single `BOOLEAN` result.

- `A AND B` is `TRUE` only when *both* `A` and `B` are `TRUE`.
- `A OR B` is `TRUE` when *at least one* of `A` or `B` is `TRUE` (including when both are `TRUE`).
- `NOT A` reverses a single `BOOLEAN` value: `TRUE` becomes `FALSE` and `FALSE` becomes `TRUE`.

Logical operators let a single `IF` statement test several conditions at once, rather than needing nested `IF` statements for every combination.

VN **GIẢI THÍCH TIẾNG VIỆT**

Các phép toán logic: `AND` chỉ cho kết quả `TRUE` khi *cả hai* điều kiện đều đúng; `OR` cho kết quả `TRUE` khi *ít nhất một trong hai* điều kiện đúng; `NOT` đảo ngược một giá trị BOOLEAN duy nhất (TRUE thành FALSE và ngược lại). Nhờ các phép logic, một câu lệnh `IF` có thể kiểm tra nhiều điều kiện cùng lúc thay vì phải lồng nhiều câu lệnh `IF` riêng biệt.

Ví dụ mẫu · Worked example

Given `DECLARE Age : INTEGER` with `Age <- 16` and `DECLARE HasTicket : BOOLEAN` with `HasTicket <- TRUE`, evaluate the following two compound conditions.

```
IF (Age >= 18) OR (HasTicket = TRUE) THEN
   OUTPUT "Condition 1 is TRUE"
ENDIF
IF (Age >= 12) AND NOT (HasTicket = FALSE) THEN
   OUTPUT "Condition 2 is TRUE"
ENDIF
```

Condition 1: `Age >= 18` is $16 \geq 18$, which is `FALSE`. `HasTicket = TRUE` is `TRUE`. `FALSE OR TRUE` evaluates to `TRUE`, so "Condition 1 is TRUE" is output.

Condition 2: `Age >= 12` is $16 \geq 12$, which is `TRUE`. `HasTicket = FALSE` is `FALSE` (since `HasTicket` is `TRUE`), so `NOT (HasTicket = FALSE)` is `NOT FALSE` $=$ `TRUE`. `TRUE AND TRUE` evaluates to `TRUE`, so "Condition 2 is TRUE" is also output.

8.3 Programming constructs

Every algorithm, however complex, is built entirely from just three basic constructs: **sequence**, **selection** and **iteration**. Combining and nesting these three constructs is sufficient to express any algorithm.

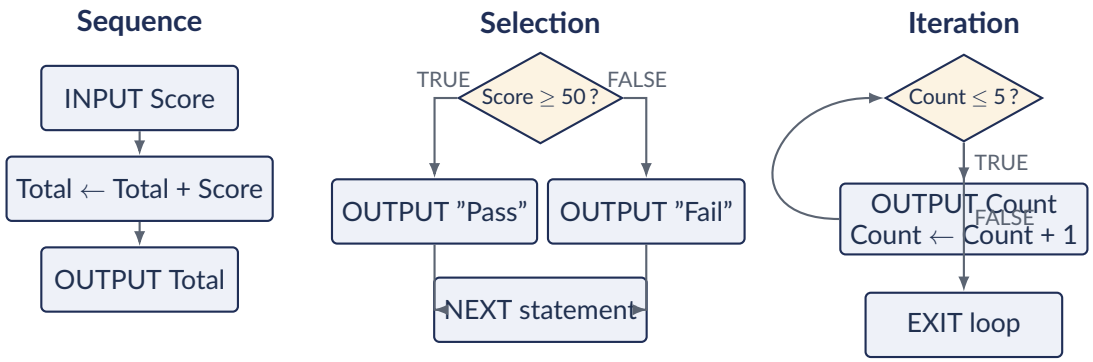


Figure 8.1: The three basic programming constructs — sequence, selection and iteration.

8.3.1 Sequence

Khái niệm cốt lõi

Sequence is the simplest construct: a set of instructions executed strictly one after another, in the exact order they are written, with no branching and no repetition.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tuần tự (sequence) là cấu trúc đơn giản nhất: các câu lệnh được thực thi lần lượt theo đúng thứ tự chúng được viết ra, không rẽ nhánh và không lặp lại.

8.3.2 Selection

Khái niệm cốt lõi

Selection allows a program to choose between different courses of action depending on whether a condition is `TRUE` or `FALSE`.

```
IF <condition> THEN
   <statements executed if condition is TRUE>
ELSE
   <statements executed if condition is FALSE>
ENDIF
```

The `ELSE` branch is optional. When there are several distinct, discrete values to choose between (rather than a range), `CASE OF` is often clearer than a chain of nested `IF` statements:

```
CASE OF <identifier>
   <value 1> : <statement(s)>
   <value 2> : <statement(s)>
   OTHERWISE : <statement(s)>
ENDCASE
```

`OTHERWISE` is optional and catches any value not explicitly listed. A `CASE OF` branch may also match a *range* of values using `TO`, e.g. `90 TO 100 : OUTPUT "Distinction"`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Rẽ nhánh (selection) cho phép chương trình chọn giữa các hướng xử lý khác nhau tuỳ theo một điều kiện là `TRUE` hay `FALSE`, dùng cấu trúc `IF...THEN...ELSE...ENDIF`. Nhánh `ELSE` là tuỳ chọn. Khi cần chọn giữa nhiều giá trị rời rạc cụ thể (thay vì một khoảng điều kiện), cấu

trúc `CASE OF...ENDCASE` thường rõ ràng hơn nhiều lệnh `IF` lồng nhau; `OTHERWISE` xử lý mọi giá trị không được liệt kê, và một nhánh của `CASE OF` cũng có thể khớp với cả một *khoảng* giá trị bằng từ khoá `TO`.

Ví dụ mẫu · Worked example

Write pseudocode using `CASE OF` that converts a numeric day-of-week (1 to 7) into its day name, and trace it for `Day <- 3`.

```
DECLARE Day : INTEGER
Day <- 3
CASE OF Day
   1 : OUTPUT "Monday"
   2 : OUTPUT "Tuesday"
   3 : OUTPUT "Wednesday"
   4 : OUTPUT "Thursday"
   5 : OUTPUT "Friday"
   6 : OUTPUT "Saturday"
   7 : OUTPUT "Sunday"
   OTHERWISE : OUTPUT "Invalid day"
ENDCASE
```

Tracing: `Day` holds the value 3. The `CASE OF` statement compares `Day` against each listed value in turn; it matches the branch `3`, so exactly one line is executed: the program outputs **Wednesday**. No other branch (including `OTHERWISE`) is evaluated once a match is found.

8.3.3 Iteration

Khái niệm cốt lõi

Iteration (looping) repeats a block of statements. Cambridge pseudocode provides three loop constructs:

- **FOR...TO...NEXT** — a **count-controlled** loop, used when the exact number of repetitions is known in advance.

```
FOR Count <- 1 TO 5
   OUTPUT Count
NEXT Count
```

- **WHILE...DO...ENDWHILE** — a **pre-condition** loop: the condition is tested *before* each pass, so the loop body may execute **zero or more** times.

```
WHILE <condition> DO
   <statements>
ENDWHILE
```

- **REPEAT...UNTIL** — a **post-condition** loop: the condition is tested *after* each pass, so the loop body always executes **at least once**.

```
REPEAT
   <statements>
UNTIL <condition>
```

VN **GIẢI THÍCH TIẾNG VIỆT**

Lặp (iteration) lặp lại một khối lệnh. `FOR...TO...NEXT` là vòng lặp **đếm trước**, dùng khi biết chính xác số lần lặp. `WHILE...DO...ENDWHILE` kiểm tra điều kiện *trước* mỗi lần lặp nên thân vòng lặp có thể chạy **0 lần hoặc nhiều lần**. `REPEAT...UNTIL` kiểm tra điều kiện *sau* mỗi lần lặp nên thân vòng lặp luôn chạy **ít nhất 1 lần**, dù điều kiện dừng đúng ngay từ đầu.

(!) Lỗi thường gặp: `WHILE` and `REPEAT...UNTIL` are *not* interchangeable just by rewriting the condition. If a condition for stopping the loop is already true *before* the loop is ever reached, a `WHILE` loop's body will run **zero times**, but a `REPEAT...UNTIL` loop's body will still run **once** before the exit condition is even checked. This matters most for input-validation loops: a `REPEAT...UNTIL` is normally the correct choice there, precisely because the user must be asked for input at least once.

Ví dụ mẫu · Worked example

Trace this `WHILE` loop that sums numbers entered by the user until a sentinel value of -1 is entered, given the inputs $5, 8, 3, -1$ in that order.

```
DECLARE Number, Total : INTEGER
Total <- 0
INPUT Number
WHILE Number <> -1 DO
   Total <- Total + Number
   INPUT Number
ENDWHILE
OUTPUT Total
```

Number (input)	Number <> −1 ?	Total
–	–	0
5	TRUE	$0 + 5 = 5$
8	TRUE	$5 + 8 = 13$
3	TRUE	$13 + 3 = 16$
-1	FALSE (loop exits)	16

Trace of a sentinel-controlled WHILE loop.

The program outputs **16**. Because the condition is checked *before* `INPUT Number` adds a value, the sentinel -1 itself is never added to `Total`.

8.4 String handling

Text data is rarely used exactly as it is typed in — programs routinely need to measure it, pull out part of it, join pieces together, or normalise its case. Cambridge pseudocode provides a small set of built-in string operations for exactly this.

Khái niệm cốt lõi

Operation	Meaning	Example
`LENGTH(str)`	Returns the number of characters in `str`	`LENGTH("HELLO")` $= 5$
`SUBSTRING(str, x, y)`	Returns `y` characters from `str`, starting at position `x`	`SUBSTRING("HELLO",2,3)` $=$ `"ELL"`
`str1 & str2`	Concatenation — joins two strings together	`"HELLO" & " WORLD"` $=$ `"HELLO WORLD"`
`UCASE(str)`	Converts every letter in `str` to upper case	`UCASE("hello")` $=$ `"HELLO"`
`LCASE(str)`	Converts every letter in `str` to lower case	`LCASE("HELLO")` $=$ `"hello"`

Built-in string handling operations in Cambridge pseudocode.

Positions within a string are counted from 1, not 0: in `"HELLO"`, position 1 is `'H'` and position 5 is `'O'`.

VN GIẢI THÍCH TIẾNG VIỆT

Cambridge cung cấp sẵn một số phép xử lý chuỗi: `LENGTH(str)` trả về số ký tự trong chuỗi; `SUBSTRING(str, x, y)` trích ra `y` ký tự bắt đầu từ vị trí `x`; `&` nối hai chuỗi lại với nhau; `UCASE(str)` chuyển toàn bộ chữ cái sang chữ hoa; `LCASE(str)` chuyển sang chữ thường. Vị trí trong chuỗi được đếm từ 1, không phải từ 0.

Ví dụ mẫu · Worked example

Given `FullName <- "ADA LOVELACE"`, where the space character sits at position 4, write pseudocode that combines `SUBSTRING`, `LENGTH`, `UCASE`, `LCASE` and concatenation (`&`) to produce a neatly formatted display name of the form `"A. Lovelace"`.

```
DECLARE FullName, Surname, FormattedSurname : STRING
DECLARE FirstInitial, DisplayName : STRING
DECLARE SpacePos : INTEGER
```

```
FullName <- "ADA LOVELACE"
SpacePos <- 4
FirstInitial <- SUBSTRING(FullName, 1, 1)
Surname <- SUBSTRING(FullName, SpacePos + 1, LENGTH(FullName) - SpacePos)
FormattedSurname <- UCASE(SUBSTRING(Surname,1,1)) & LCASE(SUBSTRING(Surname,2,LENGTH(Surnam
DisplayName <- FirstInitial & ". " & FormattedSurname
OUTPUT DisplayName
```

Tracing step by step: `LENGTH(FullName)` = 12 (`"ADA LOVELACE"` has 12 characters). `FirstInitial <- SUBSTRING(FullName,1,1)` takes 1 character from position 1, giving `"A"`. `Surname <- SUBSTRING(FullName, 5, 12-4)` = `SUBSTRING(FullName,5,8)`, which takes 8 characters starting at position 5: `"LOVELACE"`. `LENGTH(Surname)` = 8. `SUBSTRING(Surname,1,1)` = `"L"`, and `UCASE("L")="L"`. `SUBSTRING(Surname,2,7)` takes 7 characters from position 2 of `"LOVELACE"`: `"OVELACE"`, and `LCASE("OVELACE")="ovelace"`. So `FormattedSurname <- "L" & "ovelace" = "Lovelace"`. Finally `DisplayName <- "A" & ". " & "Lovelace"`, which outputs **A. Lovelace**.

(!) Lỗi thường gặp: `SUBSTRING` takes a *start position* and a *length* (how many characters to take), *not* a start position and an end position. `SUBSTRING("HELLO", 2, 3)` means "start at position 2 and take 3 characters" = `"ELL"`, not "take characters from position 2 to position 3". Mixing these two interpretations up is one of the most common string-handling errors.

8.5 Arrays

A single variable can only ever hold one value at a time. When a program needs to store many related values of the same type — five exam scores, thirty students' names, a whole grid of sensor readings — an array is used instead.

Khái niệm cốt lõi

An **array** is a fixed-size, ordered collection of data items, all of the *same* data type, stored under a single identifier and accessed using an **index** (position number). A one-dimensional array is declared:

```
DECLARE <identifier> : ARRAY[<lower>:<upper>] OF <data type>
```

For example, `DECLARE Scores : ARRAY[1:5] OF INTEGER` declares an array called `Scores` holding exactly 5 integers, indexed from 1 to 5. An individual element is accessed by writing the index in square brackets, e.g. `Scores[3]` refers to the third element.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **mảng (array)** là một tập hợp có kích thước cố định, có thứ tự, gồm các phần tử *cùng kiểu dữ liệu*, được lưu dưới một tên duy nhất và truy cập bằng **chỉ số (index)**. Mảng một chiều được khai báo theo cú pháp `DECLARE <tên> : ARRAY[<dưới>:<trên>] OF <kiểu dữ liệu>`. Ví dụ `DECLARE Scores : ARRAY[1:5] OF INTEGER` khai báo mảng `Scores` chứa đúng 5 số nguyên, đánh chỉ số từ 1 đến 5; `Scores[3]` truy cập phần tử thứ ba.

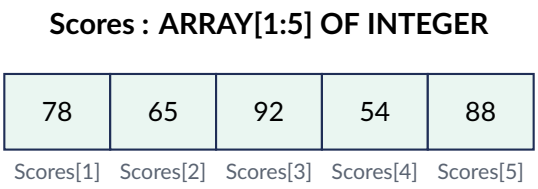


Figure 8.2: Memory representation of a one-dimensional array of five integer scores.

Ví dụ mẫu · Worked example

Write pseudocode to input 5 exam scores into an array and output their average.

```
DECLARE Scores : ARRAY[1:5] OF INTEGER
DECLARE Index, Total : INTEGER
DECLARE Average : REAL
Total <- 0
FOR Index <- 1 TO 5
   INPUT Scores[Index]
   Total <- Total + Scores[Index]
NEXT Index
Average <- Total / 5
OUTPUT Average
```

The `FOR` loop runs exactly 5 times, once for each valid index. On each pass, one score is read directly into the array element `Scores[Index]` and immediately added to the running `Total`. After the loop, dividing `Total` by 5 (using `/`, since the average may not be a whole number) gives the average, which is output.

Khái niệm cốt lõi

A **two-dimensional array** stores data arranged in **rows** and **columns**, and is declared with two index ranges:

```
DECLARE <identifier> : ARRAY[<r1>:<r2>, <c1>:<c2>] OF <data type>
```

For example, `DECLARE Grid : ARRAY[1:3, 1:4] OF INTEGER` declares a grid of 3 rows and 4 columns (12 elements in total). An individual element is accessed with *two* indices, e.g. `Grid[2,3]` refers to the element in row 2, column 3. Processing every element of a 2D array typically requires *nested* `FOR` loops — one for the row index, one (nested inside) for the column index.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **mảng hai chiều** lưu dữ liệu theo **hàng** và **cột**, khai báo với hai khoảng chỉ số: `DECLARE <tên> : ARRAY[<hàng1>:<hàng2>, <cột1>:<cột2>] OF <kiểu>`. Truy cập một phần tử cần *hai* chỉ số, ví dụ `Grid[2,3]` là phần tử ở hàng 2, cột 3. Để xử lý toàn bộ mảng hai chiều, ta thường cần *lồng hai vòng lặp* `FOR` — một vòng cho chỉ số hàng, một vòng lồng bên trong cho chỉ số cột.

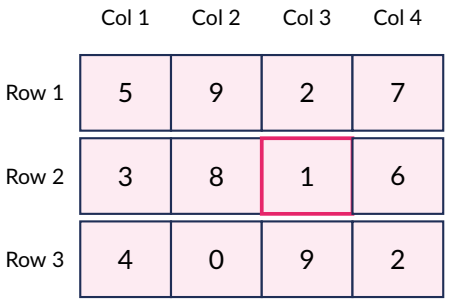


Figure 8.3: A two-dimensional array `Grid : ARRAY[1:3, 1:4] OF INTEGER`; the highlighted cell is `Grid[2,3] = 1`.

Ví dụ mẫu · Worked example

Write pseudocode using nested `FOR` loops to output every element of the 2-row, 3-column array `Grid` shown below, one row per line.

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
DECLARE Row, Col : INTEGER
FOR Row <- 1 TO 2
   FOR Col <- 1 TO 3
      OUTPUT Grid[Row, Col]
   NEXT Col
NEXT Row
```

For each value of `Row` (outer loop, 1 to 2), the inner loop runs `Col` from 1 to 3, outputting `Grid[Row,Col]` each time — so the whole first row (`Grid[1,1]`, `Grid[1,2]`, `Grid[1,3]`) is output before the outer loop moves on to `Row <- 2` and the whole second row is output. The inner loop therefore executes a total of $2 \times 3 = 6$ times.

8.5.1 Linear search

Arrays are often searched to find out whether a particular value is present, and if so, where.

Khái niệm cốt lõi

A **linear search** checks each element of an array in turn, starting from the first, comparing it against the target value, and stops as soon as a match is found (or the whole array has been checked with no match). A `BOOLEAN` **flag** variable is commonly used to record whether the target has been found, so the loop can stop early.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tìm kiếm tuần tự (linear search) kiểm tra từng phần tử của mảng lần lượt từ đầu, so sánh với giá trị cần tìm, và dừng lại ngay khi tìm thấy (hoặc khi đã kiểm tra hết mảng mà không tìm thấy). Một biến **cờ (flag)** kiểu `BOOLEAN` thường được dùng để ghi lại việc đã tìm thấy hay chưa, giúp vòng lặp có thể dừng sớm.

Ví dụ mẫu · Worked example

Write pseudocode for a linear search over a 6-element array of integers, and trace it for the array `Numbers = [12, 45, 7, 89, 34, 23]` searching first for the target value 89 (present), then for 100 (not present).

```
PROCEDURE LinearSearch(Numbers : ARRAY[1:6] OF INTEGER, Target : INTEGER)
```

```
   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN
   Found <- FALSE
   Index <- 1
   WHILE Index <= 6 AND Found = FALSE DO
      IF Numbers[Index] = Target THEN
         Found <- TRUE
         OUTPUT "Value found at position ", Index
      ENDIF
      Index <- Index + 1
   ENDWHILE
   IF Found = FALSE THEN
      OUTPUT "Value not found"
   ENDIF
ENDPROCEDURE
```

Trace for Target $= 89$**:**

Index	Numbers[Index]	Match ?	Found	Action
1	12	No	FALSE	Index → 2
2	45	No	FALSE	Index → 3
3	7	No	FALSE	Index → 4
4	89	Yes	TRUE	Output "Value found at position 4"; loop condition now false, exit

Linear search trace: target present.

Output: **Value found at position 4**.

Trace for Target $= 100$**:** every element (12, 45, 7, 89, 34, 23) is checked in turn and none equals 100, so `Found` remains `FALSE` throughout. After `Index` reaches 7 (having checked all 6 elements), the loop condition `Index <= 6` becomes false and the loop exits. Since `Found = FALSE`, the final `IF` outputs **Value not found**.

(!) Lỗi thường gặp: Cambridge array indices normally start at `1` (as declared, e.g. `ARRAY[1:6]`), *not* at `0` as in many programming languages. A loop written as `FOR Index <- 0 TO 5` on an array declared `ARRAY[1:6]` would access the non-existent element `Numbers[0]` and miss the final valid element `Numbers[6]` entirely. Always match loop bounds exactly to the array's declared lower and upper index.

8.6 Procedures, functions and parameter passing

As programs grow, repeating the same block of code in several places becomes hard to maintain and easy to get wrong. Procedures and functions let a block of code be written once and then reused, wherever it is needed, simply by calling it by name.

Khái niệm cốt lõi

A **procedure** is a named, self-contained block of code that performs a task. It is invoked using `CALL` and *may or may not* return a value; if it needs input data, that data is supplied through **parameters**.

```
PROCEDURE <name>(<parameter list>)
```

```
   <statements>
ENDPROCEDURE
...
CALL <name>(<arguments>)
```

A **function** is also a named, self-contained block of code, but it *always* returns **exactly one** value, using `RETURN`, and its declared return type is stated with `RETURNS`. Because a function always produces a value, it can be used directly inside an expression (e.g. on the right-hand side of an assignment), unlike a procedure.

```
FUNCTION <name>(<parameter list>) RETURNS <data type>
   <statements>
   RETURN <value>
ENDFUNCTION
```

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **thủ tục (procedure)** là một khối lệnh có tên, được gọi bằng `CALL`, *có thể* trả về giá trị hoặc *không*. Một **hàm (function)** cũng là một khối lệnh có tên, nhưng *luôn luôn* trả về đúng **một** giá trị bằng `RETURN`, và kiểu trả về được khai báo bằng `RETURNS`. Vì hàm luôn tạo ra một giá trị, nó có thể được dùng trực tiếp trong một biểu thức (ví dụ ở vế phải của phép gán), còn thủ tục thì không.

Ví dụ mẫu · Worked example

Write a function `IsEven` that takes an integer and returns `TRUE` if it is even, `FALSE` otherwise, and use it inside an `IF` statement.

```
FUNCTION IsEven(Number : INTEGER) RETURNS BOOLEAN
   IF Number MOD 2 = 0 THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   ENDIF
ENDFUNCTION
...
IF IsEven(8) THEN
   OUTPUT "8 is even"
ENDIF
```

`8 MOD 2 = 0`, so the `IF` inside the function is `TRUE` and `IsEven` returns `TRUE`. Because `IsEven(8)` evaluates to a `BOOLEAN` value, it can be used directly as the condition of the outer `IF` statement — this is only possible because `IsEven` is a *function* (it always returns a value); a *procedure* could not be used this way.

8.6.1 Parameter passing: by value vs. by reference

When an argument is passed into a procedure or function, Cambridge pseudocode distinguishes *how* it is passed, using `BYVAL` or `BYREF` before the parameter's data type.

Khái niệm cốt lõi

Passing by value (`BYVAL`): the procedure or function receives a **copy** of the argument's value in a brand-new piece of memory. Any change made to the parameter *inside* the procedure/function has **no effect** on the original variable in the calling code, because they occupy different memory locations.

Passing by reference (`BYREF`): the procedure or function receives a **reference to the original variable's memory location** — no copy is made. Any change made to the parameter *inside* the procedure/function **does change** the original variable, because the parameter and the original variable are literally the same memory location under two names.

VN **GIẢI THÍCH TIẾNG VIỆT**

Truyền theo giá trị (`BYVAL`): thủ tục/hàm nhận một **bản sao** của giá trị đối số, trong một ô nhớ hoàn toàn mới. Mọi thay đổi bên trong thủ tục/hàm **không** ảnh hưởng đến biến gốc ở nơi gọi, vì chúng nằm ở hai ô nhớ khác nhau. **Truyền theo tham chiếu (`BYREF`)**: thủ tục/hàm nhận một **tham chiếu đến chính ô nhớ của biến gốc** — không có bản sao nào được tạo ra. Mọi thay đổi bên trong thủ tục/hàm **sẽ** làm thay đổi biến gốc, vì tham số và biến gốc thực chất là cùng một ô nhớ, chỉ mang hai cái tên.

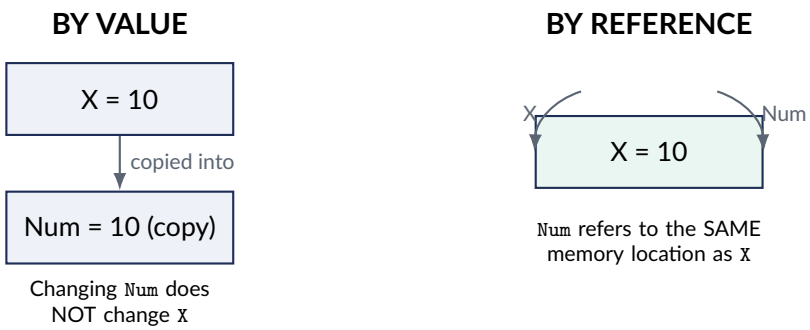


Figure 8.4: Parameter passing by value (a separate copy) versus by reference (the same memory location).

Ví dụ mẫu · Worked example

Trace the following program, which calls one procedure by value and another (identical apart from parameter passing) by reference, starting from `X <- 10`.

```
PROCEDURE ChangeValue(BYVAL Num : INTEGER)
   Num <- Num * 2
   OUTPUT "Inside procedure (by value): ", Num
ENDPROCEDURE

PROCEDURE ChangeValueRef(BYREF Num : INTEGER)
   Num <- Num * 2
   OUTPUT "Inside procedure (by reference): ", Num
ENDPROCEDURE

DECLARE X : INTEGER
X <- 10
CALL ChangeValue(X)
OUTPUT "After by-value call, X = ", X
CALL ChangeValueRef(X)
```

```
OUTPUT "After by-reference call, X = ", X
```

Step	Values
`X <- 10`	X = 10
`CALL ChangeValue(X)`: `Num` receives a *copy* of X (10); `Num <- Num*2` makes `Num` = 20 inside the procedure only	Output: "Inside procedure (by value): 20"; X is still 10
`OUTPUT "After by-value call, X = ", X`	Output: "After by-value call, X = 10"
`CALL ChangeValueRef(X)`: `Num` refers to the *same* memory as X; `Num <- Num*2` changes X itself to 20	Output: "Inside procedure (by reference): 20"; X is now 20
`OUTPUT "After by-reference call, X = ", X`	Output: "After by-reference call, X = 20"

Trace showing X unchanged after a by-value call, but changed after a by-reference call.

(!) Lỗi thường gặp: Do not assume that changing a parameter inside a procedure automatically changes the caller's variable. This is only true when the parameter is declared `BYREF`. If a parameter is declared `BYVAL` (or no passing mechanism is specified and the default behaviour applies), the procedure is only ever working on its own private copy, and the original variable in the calling code is completely unaffected once the procedure ends.

8.6.2 Variable scope

Khái niệm cốt lõi

The **scope** of a variable is the part of the program in which it can be referred to. A variable declared *inside* a procedure or function is **local** — it exists only for the duration of that procedure/function call and cannot be accessed anywhere else, including in other procedures or in the main program. A variable declared in the main body of the program (outside any procedure/function) is **global** — it can be accessed from anywhere in the program, including inside procedures and functions.

VN **GIẢI THÍCH TIẾNG VIỆT**

Phạm vi (scope) của một biến là phần chương trình mà biến đó có thể được truy cập. Biến khai báo *bên trong* một thủ tục/hàm là biến **cục bộ (local)** — chỉ tồn tại trong thời gian thủ tục/hàm đó đang chạy, không thể truy cập từ nơi khác, kể cả từ thủ tục khác. Biến khai báo ở phần thân chính của chương trình là biến **toàn cục (global)** — có thể truy cập từ bất kỳ đâu trong chương trình, kể cả từ bên trong các thủ tục/hàm.

8.7 File handling

Data stored only in variables disappears the moment a program ends. To keep data permanently, a program reads from and writes to files stored on disk.

Khái niệm cốt lõi

Working with a text file always follows the same three-stage pattern: **open** the file, **process** it (read and/or write, usually inside a loop), then **close** it.

- `OPENFILE <filename> FOR <mode>` — opens the file. Common modes are `READ` (to read existing data), `WRITE` (to write new data — this **overwrites** any existing content in the file), and `APPEND` (to add new data onto the *end* of the file, keeping what was already there).
- `READFILE <filename>, <variable>` — reads the next line from the file into `<variable>`.
- `WRITEFILE <filename>, <data>` — writes `<data>` as a new line in the file.
- `EOF(<filename>)` — returns `TRUE` once the **end of file** has been reached (no more lines left to read); used to control a reading loop.
- `CLOSEFILE <filename>` — closes the file, releasing it and ensuring any data written is properly saved.

VN **GIẢI THÍCH TIẾNG VIỆT**

Làm việc với tệp văn bản luôn theo ba bước: **mở tệp**, **xử lý** (đọc và/hoặc ghi, thường trong vòng lặp), rồi **đóng tệp**. `OPENFILE ... FOR READ/WRITE/APPEND` mở tệp ở chế độ tương ứng — `WRITE` sẽ **ghi đè**, xoá hết nội dung cũ, còn `APPEND` sẽ **thêm vào cuối** tệp mà không xoá nội dung cũ. `READFILE` đọc một dòng vào biến; `WRITEFILE` ghi một dòng dữ liệu mới; `EOF(...)` trả về `TRUE` khi đã đọc đến cuối tệp; `CLOSEFILE` đóng tệp, đảm bảo dữ liệu ghi được lưu lại đúng cách.

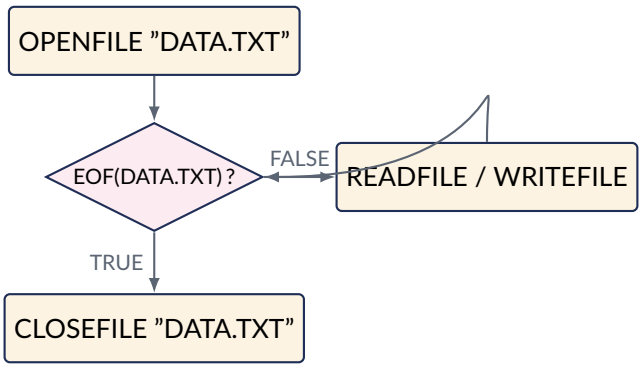


Figure 8.5: The general file-handling pattern — open, loop reading or writing until the end of file, then close.

Ví dụ mẫu · Worked example

Write pseudocode to open a text file, read and output every line it contains, then close it.

```
DECLARE Line : STRING
OPENFILE "NOTES.TXT" FOR READ
WHILE NOT EOF("NOTES.TXT") DO
   READFILE "NOTES.TXT", Line
   OUTPUT Line
ENDWHILE
CLOSEFILE "NOTES.TXT"
```

The `WHILE` loop is pre-condition controlled: `EOF` is checked *before* each `READFILE`, so if the file happened to be completely empty, no line would be read at all and no error would occur. Each pass reads exactly one line into `Line` and outputs it, continuing until the end of the file is reached, after which `CLOSEFILE` releases the file.

Ví dụ mẫu · Worked example

Write pseudocode to write three students' names and scores, one per line, to a new file called `RESULTS.TXT`, overwriting any previous content.

```
DECLARE Names  : ARRAY[1:3] OF STRING
DECLARE Scores : ARRAY[1:3] OF INTEGER
DECLARE Index  : INTEGER
Names[1] <- "Alice"
Names[2] <- "Ben"
Names[3] <- "Cara"
Scores[1] <- 82
Scores[2] <- 67
Scores[3] <- 91
OPENFILE "RESULTS.TXT" FOR WRITE
FOR Index <- 1 TO 3
   WRITEFILE "RESULTS.TXT", Names[Index] & "," & Scores[Index]
NEXT Index
CLOSEFILE "RESULTS.TXT"
```

Opening `FOR WRITE` means the file starts empty, so all three lines written are the only content in `RESULTS.TXT` once the program finishes: `"Alice,82"`, `"Ben,67"`, `"Cara,91"` (one per line, in that order). Had the file instead been opened `FOR APPEND`, these three lines would have been added *after* whatever content the file already contained, rather than replacing it.

(!) Lỗi thường gặp: Always call `CLOSEFILE` once processing is finished. If a program finishes without closing a file, changes written to it may not be properly saved to disk, and the file may remain locked so that other parts of the program (or other programs) cannot open it. Every `OPENFILE` should be matched by exactly one corresponding `CLOSEFILE`.

8.8 Practice

Bài tập luyện tập

A program needs to store a student's exam percentage, which may include a decimal fraction (e.g. 87.5%). Which data type should be used?

(A) INTEGER
(B) REAL
(C) CHAR
(D) BOOLEAN

Lời giải chi tiết

A percentage with a possible decimal part (such as 87.5) cannot be stored exactly as a whole number, so `INTEGER` would lose the fractional part. `REAL` stores numbers that may include a decimal component. **(B)**.

Bài tập luyện tập

Trace the following pseudocode fully and state the value output.

```
DECLARE Grid : ARRAY[1:2, 1:3] OF INTEGER
```

```
DECLARE Row, Col, Total : INTEGER
Grid[1,1] <- 4
Grid[1,2] <- 1
Grid[1,3] <- 2
Grid[2,1] <- 3
Grid[2,2] <- 2
Grid[2,3] <- 2
Total <- 0
FOR Row <- 1 TO 2
   FOR Col <- 1 TO 3
      Total <- Total + Grid[Row, Col]
   NEXT Col
NEXT Row
OUTPUT Total
```

(A) 10
(B) 12
(C) 14
(D) 15

Lời giải chi tiết

Row	Col	Grid[Row,Col]	Total
(before loop)	—	—	0
1	1	4	4
1	2	1	5
1	3	2	7
2	1	3	10
2	2	2	12
2	3	2	14

Trace of the nested-loop 2D array summation.

The outer loop runs `Row` from 1 to 2; for each `Row`, the inner loop runs `Col` from 1 to 3, adding each element to `Total`. The final value output is **14**. **(C)**.

Bài tập luyện tập

Explain the difference between a procedure and a function, giving one example use of each.

Lời giải chi tiết

A **procedure** is a named block of reusable code, invoked with `CALL`, which *may or may not* return a value — its purpose is usually to *perform an action* (e.g. display a welcome message, write a record to a file). A **function** is also a named block of reusable code, but it *always* returns **exactly one** value using `RETURN`, and because it always produces a value, it can be used directly inside an expression — for example, `IsEven(Number)` could appear directly inside `IF IsEven(Number) THEN`, which is not possible with a procedure. Example procedure: `PROCEDURE DisplayMenu()` which simply outputs a list of menu options and performs no calculation that needs to be returned. Example function: `FUNCTION Square(Num : INTEGER) RETURNS INTEGER`, which calculates and `RETURN`s `Num * Num` so that the result can be used immediately, e.g. `Area <- Square(Side)`.

Bài tập luyện tập

The following pseudocode is intended to open a file, output every line, and close the file, but it contains one bug. Identify the bug and rewrite the pseudocode correctly.

```
DECLARE Line : STRING
OPENFILE "NOTES.TXT" FOR READ
WHILE NOT EOF("NOTES.TXT") DO
   READFILE "NOTES.TXT", Line
   OUTPUT Line
ENDWHILE
```

Lời giải chi tiết

The bug is a **missing** `CLOSEFILE` statement: the file `"NOTES.TXT"` is opened but never closed once the reading loop finishes. This can leave the file locked (unavailable to other parts of the program) and risks data not being properly saved if any writing had occurred. The corrected pseudocode adds `CLOSEFILE` immediately after the loop:

```
DECLARE Line : STRING
OPENFILE "NOTES.TXT" FOR READ
WHILE NOT EOF("NOTES.TXT") DO
   READFILE "NOTES.TXT", Line
   OUTPUT Line
ENDWHILE
CLOSEFILE "NOTES.TXT"
```

Bài tập luyện tập

A program contains two procedures, `ProcedureA` and `ProcedureB`. `ProcedureA` declares a local variable `DECLARE Counter : INTEGER`. Explain whether `ProcedureB` can access the value of `Counter`, and why.

Lời giải chi tiết

`ProcedureB` **cannot** access `Counter`. Because `Counter` is declared *inside* `ProcedureA`, it is a **local variable**: its scope is limited to `ProcedureA` alone, and it exists only while `ProcedureA` is executing. Once `ProcedureA` finishes running, `Counter` no longer exists at all. `ProcedureB` has no way to refer to it, whether `ProcedureB` runs before, after, or even during a call to `ProcedureA` (unless `Counter` were instead declared as a global variable, or passed explicitly as a parameter).

Bài tập luyện tập

A program uses `CONSTANT VATRate = 0.20` throughout its calculations rather than typing `0.20` directly wherever it is needed. Give two reasons why declaring it as a constant is better practice.

Lời giải chi tiết

(1) **Readability**: `VATRate` makes the purpose of the value immediately clear to anyone reading the code, whereas a bare `0.20` typed repeatedly gives no indication of what it represents (a "magic number"). (2) **Single point of update**: if the VAT rate changes (e.g. to 0.225), only the

one `CONSTANT` declaration needs editing, rather than searching through the whole program for every place `0.20` was typed directly — which is slower and risks missing an occurrence, leaving an inconsistent calculation somewhere in the code.

Bài tập luyện tập

Evaluate `29 DIV 6` and `29 MOD 6`.

(A) DIV = 4, MOD = 5
(B) DIV = 5, MOD = 4
(C) DIV = 4, MOD = 4
(D) DIV = 5, MOD = 5

Lời giải chi tiết

$29/6 = 4.8\overline{3}$, so the whole-number quotient is `29 DIV 6 = 4`. The remainder is $29 - (4 \times 6) = 29 - 24 = 5$, so `29 MOD 6 = 5`. Check: $4 \times 6 + 5 = 29$. ✓ **(A)**.

Bài tập luyện tập

Which of the following is `TRUE`?

(A) $5 <> 5$
(B) $3 >= 4$
(C) $7 <= 7$
(D) $2 > 9$

Lời giải chi tiết

$5 <> 5$ means "5 is not equal to 5", which is `FALSE`. $3 >= 4$ is `FALSE`. $2 > 9$ is `FALSE`. $7 <= 7$ ("7 is less than or equal to 7") is `TRUE`, since <= includes equality. **(C)**.

Bài tập luyện tập

Given `DECLARE Temperature : INTEGER` with `Temperature <- 5` and `DECLARE AlarmOn : BOOLEAN` with `AlarmOn <- FALSE`, evaluate:

```
IF (Temperature < 0) OR (AlarmOn = TRUE) THEN
   OUTPUT "Warning"
ELSE
   OUTPUT "Normal"
ENDIF
```

Lời giải chi tiết

`Temperature < 0` is $5 < 0$, which is `FALSE`. `AlarmOn = TRUE` is `FALSE` (since `AlarmOn` holds `FALSE`). `FALSE OR FALSE` evaluates to `FALSE`, so the `IF` condition is not met and the `ELSE` branch runs. The output is **Normal**.

Bài tập luyện tập

Trace the following `CASE OF` pseudocode for the input `Score <- 76`, and state the output.

```
DECLARE Score : INTEGER
Score <- 76
CASE OF Score
   90 TO 100 : OUTPUT "Distinction"
```

```
   70 TO 89  : OUTPUT "Merit"
   50 TO 69  : OUTPUT "Pass"
   OTHERWISE : OUTPUT "Fail"
ENDCASE
```

Lời giải chi tiết

`Score` holds 76. Checking each range in order: 76 is not in 90 to 100; 76 *is* in the range 70 to 89, so this branch matches and its statement runs. The output is **Merit**. (The remaining branches, `50 TO 69` and `OTHERWISE`, are never checked once a match is found.)

Bài tập luyện tập

Write pseudocode using `CASE OF` that converts a month number (1 to 12) into its season, using the grouping: December, January, February → "Winter"; March, April, May → "Spring"; June, July, August → "Summer"; September, October, November → "Autumn".

Lời giải chi tiết

```
DECLARE Month : INTEGER
CASE OF Month
   12, 1, 2 : OUTPUT "Winter"
   3, 4, 5  : OUTPUT "Spring"
   6, 7, 8  : OUTPUT "Summer"
   9, 10, 11 : OUTPUT "Autumn"
   OTHERWISE : OUTPUT "Invalid month"
ENDCASE
```

Each branch lists the discrete month numbers that belong to one season, separated by commas; `CASE OF` matches `Month` against these listed values and executes the single matching branch. `OTHERWISE` safely handles any value outside 1–12.

Bài tập luyện tập

Trace the following `REPEAT...UNTIL` loop, given the user enters 15, then -2, then 7 in that order, and state how many times the loop body executes and the final output.

```
DECLARE Num : INTEGER
REPEAT
   INPUT Num
UNTIL Num >= 1 AND Num <= 10
OUTPUT "Valid number entered: ", Num
```

Lời giải chi tiết

Pass	Num (input)	Num ≥ 1 AND Num ≤ 10 ?
1	15	FALSE ($15 \leq 10$ is false) — repeat
2	-2	FALSE ($-2 \geq 1$ is false) — repeat
3	7	TRUE — loop ends

Trace of a REPEAT...UNTIL input-validation loop.

Because `REPEAT...UNTIL` checks its condition *after* each pass, the body (asking for input) runs

3 times in total before a valid value is accepted. The final output is **Valid number entered: 7**.

Bài tập luyện tập

Given `FullName <- "PAUL ADAMS"`, where the space sits at position 5, trace the following pseudocode and state the value output.

```
DECLARE FullName, Surname, FormattedSurname : STRING
DECLARE FirstInitial, DisplayName : STRING
DECLARE SpacePos : INTEGER
FullName <- "PAUL ADAMS"
SpacePos <- 5
FirstInitial <- SUBSTRING(FullName, 1, 1)
Surname <- SUBSTRING(FullName, SpacePos + 1, LENGTH(FullName) - SpacePos)
FormattedSurname <- UCASE(SUBSTRING(Surname,1,1)) & LCASE(SUBSTRING(Surname,2,LENGTH(Surnar
DisplayName <- FirstInitial & ". " & FormattedSurname
OUTPUT DisplayName
```

Lời giải chi tiết

`LENGTH(FullName)` $=$ 10 (`"PAUL ADAMS"` has 10 characters). `FirstInitial <- SUBSTRING(FullName,1,1)` $=$ `"P"`. `Surname <- SUBSTRING(FullName, 6, 10-5)` $=$ `SUBSTRING(FullName,6,5)`, taking 5 characters from position 6: `"ADAMS"`. `LENGTH(Surname)` $=$ 5. `SUBSTRING(Surname,1,1)` $=$ `"A"`, `UCASE("A")` $=$ `"A"`. `SUBSTRING(Surname,2,4)` takes 4 characters from position 2 of `"ADAMS"`: `"DAMS"`, and `LCASE("DAMS")` $=$ `"dams"`. So `FormattedSurname <- "A" & "dams" = "Adams"`. Finally `DisplayName <- "P" & ". " & "Adams"`. The output is **P. Adams**.

Bài tập luyện tập

Write pseudocode using a `REPEAT...UNTIL` loop and `UCASE` that repeatedly asks the user to enter "Yes" or "No", accepting the answer regardless of the case the user types it in (e.g. "yes", "YES" and "Yes" should all be accepted).

Lời giải chi tiết

```
DECLARE Answer : STRING
DECLARE ValidInput : BOOLEAN
REPEAT
   INPUT Answer
   IF UCASE(Answer) = "YES" OR UCASE(Answer) = "NO" THEN
      ValidInput <- TRUE
   ELSE
      ValidInput <- FALSE
      OUTPUT "Please enter Yes or No"
   ENDIF
UNTIL ValidInput = TRUE
```

Converting `Answer` to upper case with `UCASE` before comparing it against `"YES"` and `"NO"` means the comparison no longer depends on how the user capitalised their answer — `UCASE("yes")`, `UCASE("YES")` and `UCASE("Yes")` all equal `"YES"`. The `REPEAT...UNTIL` loop guarantees the user is asked at least once and keeps asking until a valid answer (in either case)

is entered.

Bài tập luyện tập

The array `Marks = [56, 78, 90, 41, 65]` (indices 1–5) is searched using the `LinearSearch` procedure from the theory section. (a) Trace the search for `Target = 90`. (b) Trace the search for `Target = 100`.

Lời giải chi tiết

(a) **Target** $= 90$**:** Index 1: `Marks[1]`$= 56 \neq 90$. Index 2: `Marks[2]`$= 78 \neq 90$. Index 3: `Marks[3]`$= 90 = 90$ — match: `Found <- TRUE`, output "Value found at position 3"; the loop condition (`Found = FALSE`) is now false, so the loop exits. Final output: **Value found at position 3**.

(b) **Target** $= 100$**:** all five elements $(56, 78, 90, 41, 65)$ are checked in turn and none equals 100, so `Found` stays `FALSE`. After `Index` becomes 6, the condition `Index <= 5` is false and the loop exits. Since `Found = FALSE`, the final `IF` outputs **Value not found**.

Bài tập luyện tập

Write pseudocode for a linear search that searches a 4-element array of names, `Names : ARRAY[1:4] OF STRING`, for a `STRING` value input by the user, outputting the position if found or a "not found" message otherwise.

Lời giải chi tiết

```
DECLARE Names : ARRAY[1:4] OF STRING
DECLARE SearchName : STRING
DECLARE Index : INTEGER
DECLARE Found : BOOLEAN
INPUT SearchName
Found <- FALSE
Index <- 1
WHILE Index <= 4 AND Found = FALSE DO
   IF Names[Index] = SearchName THEN
      Found <- TRUE
      OUTPUT "Name found at position ", Index
   ENDIF
   Index <- Index + 1
ENDWHILE
IF Found = FALSE THEN
   OUTPUT "Name not found"
ENDIF
```

This follows the same structure as the numeric linear search: a `Found` flag starts `FALSE`, the loop checks each element in turn and stops early once a match is located, and a final `IF` after the loop reports "not found" only if no match was ever made.

Bài tập luyện tập

Which declaration correctly creates a two-dimensional array of integers with 3 rows and 4 columns?

(A) DECLARE Grid : ARRAY[3, 4] OF INTEGER

(B) DECLARE Grid : ARRAY[1:3] OF ARRAY[1:4] OF INTEGER

(C) DECLARE Grid : ARRAY[1:3, 1:4] OF INTEGER

(D) DECLARE Grid : ARRAY(1:3, 1:4) OF INTEGER

Lời giải chi tiết

Cambridge pseudocode declares a two-dimensional array with two index ranges inside a single pair of square brackets, separated by a comma: `ARRAY[<row lower>:<row upper>, <col lower>:<col upper>] OF <type>`. **(C)**.

Bài tập luyện tập

Trace the following program, starting from `X <- 7`, and state the value of `X` output after each of the two procedure calls.

```
PROCEDURE ChangeValue(BYVAL Num : INTEGER)
   Num <- Num * 2
ENDPROCEDURE

PROCEDURE ChangeValueRef(BYREF Num : INTEGER)
   Num <- Num * 2
ENDPROCEDURE

DECLARE X : INTEGER
X <- 7
CALL ChangeValue(X)
OUTPUT "After by-value call, X = ", X
CALL ChangeValueRef(X)
OUTPUT "After by-reference call, X = ", X
```

Lời giải chi tiết

`X <- 7`. `CALL ChangeValue(X)`: `Num` receives a *copy* of `X` (7); inside the procedure `Num` becomes 14, but this has no effect on `X`, since `Num` is declared `BYVAL`. Output: **After by-value call, X = 7**. `CALL ChangeValueRef(X)`: `Num` refers to the *same* memory location as `X`, since it is declared `BYREF`; `Num <- Num * 2` changes `X` itself from 7 to 14. Output: **After by-reference call, X = 14**.

Bài tập luyện tập

Write a procedure `Swap` that exchanges the values of two integer variables, so that after `CALL Swap(X, Y)` with `X <- 3` and `Y <- 9`, `X` holds 9 and `Y` holds 3.

Lời giải chi tiết

```
PROCEDURE Swap(BYREF A : INTEGER, BYREF B : INTEGER)
   DECLARE Temp : INTEGER
   Temp <- A
   A <- B
   B <- Temp
ENDPROCEDURE
```

Both parameters *must* be declared `BYREF`: the whole point of `Swap` is to change the caller's original variables, which is only possible if the parameters refer to the same memory locations as `X` and `Y`. Tracing `CALL Swap(X, Y)` with $X = 3, Y = 9$: `Temp <- A` sets `Temp <- 3`; `A <- B` sets `X <- 9` (since `A` is `X`); `B <- Temp` sets `Y <- 3` (since `B` is `Y`). After the call, `X = 9` and `Y = 3`, as required. If `A` and `B` had instead been declared `BYVAL`, the swap would happen only on local copies and `X` and `Y` would be completely unchanged after the call.

Bài tập luyện tập

Write pseudocode to write the names and prices of 4 products, stored in the arrays `ProductNames : ARRAY[1:4] OF STRING` and `Prices : ARRAY[1:4] OF REAL`, one per line, to a new file `INVENTORY.TXT`, overwriting any previous content.

Lời giải chi tiết

```
DECLARE ProductNames : ARRAY[1:4] OF STRING
DECLARE Prices : ARRAY[1:4] OF REAL
DECLARE Index : INTEGER
OPENFILE "INVENTORY.TXT" FOR WRITE
FOR Index <- 1 TO 4
   WRITEFILE "INVENTORY.TXT", ProductNames[Index] & "," & Prices[Index]
NEXT Index
CLOSEFILE "INVENTORY.TXT"
```

Opening `FOR WRITE` guarantees the file starts empty, so `INVENTORY.TXT` ends up containing exactly the four lines written by the loop, one product per line, in array order, with the name and price on each line joined by a comma using concatenation (`&`).

Bài tập luyện tập

A text file `WORDS.TXT` contains exactly three lines: `apple`, `Banana`, `cherry`. Trace the following pseudocode and state the three lines output.

```
DECLARE Word : STRING
OPENFILE "WORDS.TXT" FOR READ
WHILE NOT EOF("WORDS.TXT") DO
   READFILE "WORDS.TXT", Word
   OUTPUT UCASE(Word)
ENDWHILE
CLOSEFILE "WORDS.TXT"
```

Lời giải chi tiết

The loop reads and outputs each line of `WORDS.TXT` in turn, converting it to upper case with `UCASE` before output. `UCASE("apple")` = `"APPLE"`; `UCASE("Banana")` = `"BANANA"`; `UCASE("cherry")` = `"CHERRY"`. The three lines output, in order, are **APPLE**, **BANANA**, **CHERRY**. Once all three lines have been read, `EOF("WORDS.TXT")` becomes `TRUE`, the loop ends, and the file is closed.

Bài tập luyện tập

Which loop construct guarantees that its body executes **at least once**, even if the exit condition is already true before the loop begins?

(A) FOR...TO...NEXT
(B) WHILE...DO...ENDWHILE
(C) REPEAT...UNTIL
(D) CASE OF...ENDCASE

Lời giải chi tiết

`REPEAT...UNTIL` tests its condition *after* the loop body runs, so the body always executes at least once, regardless of whether the condition would already have been true beforehand. A `WHILE` loop tests its condition *before* the body runs and so may execute the body zero times. `CASE OF` is a selection construct, not a loop. **(C)**.

Bài tập luyện tập

Write a function `IsVowel` that takes a single `CHAR` and returns `TRUE` if it is a vowel (A, E, I, O or U, upper or lower case) and `FALSE` otherwise. Trace `IsVowel('e')` and `IsVowel('z')`.

Lời giải chi tiết

```
FUNCTION IsVowel(Letter : CHAR) RETURNS BOOLEAN
   DECLARE UpperLetter : STRING
   UpperLetter <- UCASE(Letter)
   IF UpperLetter = "A" OR UpperLetter = "E" OR UpperLetter = "I" OR UpperLetter = "O" OR U
      RETURN TRUE
   ELSE
      RETURN FALSE
   ENDIF
ENDFUNCTION
```

`IsVowel('e')`: `UCASE('e')` = `"E"`, which matches the second condition in the `OR` chain, so the function returns **TRUE**. `IsVowel('z')`: `UCASE('z')` = `"Z"`, which matches none of `"A"`, `"E"`, `"I"`, `"O"`, `"U"`, so the `ELSE` branch runs and the function returns **FALSE**. Converting `Letter` to upper case first means the function correctly handles both upper- and lower-case input with a single comparison chain.

Bài tập luyện tập

The pseudocode below is intended to input 4 numbers into an array and output their total, but it contains three separate errors. Identify all three and rewrite the pseudocode correctly.

```
DECLARE Numbers : ARRAY[1:4] OF INTEGER
DECLARE Index, Total : INTEGER
```

```
Total <- 0
FOR Index <- 0 TO 4
   INPUT Numbers[Index]
   Total <- Total + Numbers[Index]
NEXT Index
OUTPUT Total
```

Lời giải chi tiết

Three errors: (1) The loop starts at `Index <- 0`, but the array is declared `ARRAY[1:4]`, so valid indices only run from 1 to 4 — `Numbers[0]` does not exist. (2) The loop's upper bound is `4`, but since it starts at `0`, it runs 5 times (`Index = 0,1,2,3,4`) instead of exactly 4, one too many. (3) The loop bound should match the array declaration exactly (`1 TO 4`), not `0 TO 4`. Corrected pseudocode:

```
DECLARE Numbers : ARRAY[1:4] OF INTEGER
DECLARE Index, Total : INTEGER
Total <- 0
FOR Index <- 1 TO 4
   INPUT Numbers[Index]
   Total <- Total + Numbers[Index]
NEXT Index
OUTPUT Total
```

The loop now correctly runs `Index` from 1 to 4, matching the array's declared bounds exactly, reading and totalling exactly the 4 elements that actually exist.

Bài tập luyện tập

Trace the following `WHILE` loop, which sums only the **even** numbers entered by the user until a sentinel value of 0 is entered, given the inputs $4, 7, 10, 0$ in that order. State the value output.

```
DECLARE Number, EvenTotal : INTEGER
EvenTotal <- 0
INPUT Number
WHILE Number <> 0 DO
   IF Number MOD 2 = 0 THEN
      EvenTotal <- EvenTotal + Number
   ENDIF
   INPUT Number
ENDWHILE
OUTPUT EvenTotal
```

Lời giải chi tiết

Number (input)	Number $<>0$?	Number MOD 2 = 0 ?	EvenTotal
—	—	—	0
4	TRUE	TRUE (even)	$0+4=4$
7	TRUE	FALSE (odd)	4 (unchanged)
10	TRUE	TRUE (even)	$4+10=14$
0	FALSE (loop exits)	—	14

Trace of a WHILE loop that totals only even sentinel-terminated inputs.

The condition is checked *before* each pass, so once `Number` becomes 0 the loop exits immediately without adding it. Only 4 and 10 pass the `MOD 2 = 0` test, so `EvenTotal` $= 4 + 10 = 14$. The program outputs **14**.

Bài tập luyện tập

Given `DECLARE Age : INTEGER` with `Age <- 20` and `DECLARE HasID : BOOLEAN` with `HasID <- FALSE`, what does the following pseudocode output?

```
IF (Age >= 18) AND NOT (HasID = TRUE) THEN
   OUTPUT "Eligible"
ELSE
   OUTPUT "Not eligible"
ENDIF
```

(A) Eligible

(B) Not eligible

(C) Error — HasID must be TRUE

(D) Cannot be determined

Lời giải chi tiết

`Age >= 18` is $20 \geq 18$, which is `TRUE`. `HasID = TRUE` is `FALSE` (since `HasID` holds `FALSE`), so `NOT(HasID = TRUE)` is `NOT FALSE` = `TRUE`. `TRUE AND TRUE` evaluates to `TRUE`, so the `IF` branch runs and the program outputs **Eligible**. **(A)**.

Databases

Mục tiêu Unit: Khái niệm cơ sở dữ liệu (database), bảng (table), bản ghi (record) và trường dữ liệu (field); các kiểu dữ liệu trường phổ biến; khoá chính (primary key); lý do sử dụng cơ sở dữ liệu thay vì các tệp rời rạc; kiểm tra hợp lệ (validation) áp dụng cho dữ liệu cơ sở dữ liệu; truy vấn SQL với `SELECT...FROM...WHERE...ORDER BY`, các phép so sánh, kết hợp điều kiện với `AND/OR`, và ký tự đại diện `LIKE` với `%`; biểu mẫu (form) và báo cáo (report).

9.1 Database concepts

Almost every piece of software that stores information about people, products, bookings or transactions is built on top of a **database**. Before any SQL can be written, the basic vocabulary of how a database is structured must be completely secure.

Khái niệm cốt lõi

A **database** is an organised collection of related, structured data, stored so that it can be efficiently entered, searched, sorted, updated and retrieved.

- A database is made up of one or more **tables**. A table stores data about one particular type of "thing" (for example, students, books, or orders).
- Each table is made up of **records** (rows). One record holds all the data about a single item — one student, one book, one order.
- Each record is made up of **fields** (columns). A field holds one single piece of information about the record — for example, a student's first name, or a book's year of publication. Every field has a **data type** that fixes what kind of value it may hold (Text, Number, Date, Boolean, Currency, and so on).
- A **primary key** is a field (or combination of fields) chosen so that its value is **guaranteed to be unique and never blank** for every single record in the table. The primary key is what the database uses internally to identify one exact record with no ambiguity.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một **cơ sở dữ liệu (database)** là một tập hợp dữ liệu có tổ chức, có cấu trúc và có liên quan với nhau, được lưu trữ để có thể nhập, tìm kiếm, sắp xếp, cập nhật và truy xuất một cách hiệu quả. Một cơ sở dữ liệu gồm một hoặc nhiều **bảng (table)**; mỗi bảng gồm nhiều **bản ghi (record)** — tương ứng với một dòng, lưu toàn bộ thông tin về một đối tượng cụ thể (một học sinh, một cuốn sách...); mỗi bản ghi lại gồm nhiều **trường (field)** — tương ứng với một cột, mỗi trường lưu đúng một mẩu thông tin và có một **kiểu dữ liệu** cố định. **Khoá chính (primary key)** là trường (hoặc tổ hợp trường) được chọn sao cho giá trị của nó **luôn duy nhất và không bao giờ để trống** ở mọi bản ghi — đây là cách cơ sở dữ liệu xác định chính xác một bản ghi duy nhất, không nhầm lẫn.

The software that actually creates, stores and lets a user query a database is called a **Database Management System (DBMS)**. Rather than each program that needs the data having to read and

write its own private file format, every program instead sends its requests through the DBMS, which enforces the table structure, applies validation rules, and answers queries such as the SQL statements introduced later in this chapter. At this level, the databases considered are all **single-table** databases — everything needed is stored in one table, so there is no need to link separate tables together; the ideas of tables, records, fields and primary keys below apply equally, however, to every table inside a larger, multi-table database.

The table below is used throughout this chapter as the main worked example. It stores basic academic records for five students in a secondary school.

Sample table: Students

StudentID	FirstName	LastName	YearGroup	Grade
S001	Amelia	Turner	10	A
S002	Brandon	Ng	11	B
S003	Chloe	Nguyen	10	B
S004	Daniel	Osei	11	A
S005	Isabella	Novak	10	C

Field data types: StudentID = Text, FirstName = Text, LastName = Text, YearGroup = Number, Grade = Character.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng **Students** ở trên sẽ được dùng xuyên suốt chương này. **StudentID** (mã số học sinh) là **khoá chính**: mỗi mã (S001 đến S005) là duy nhất và không bao giờ để trống. **YearGroup** (khối lớp, 10 hoặc 11) có kiểu **Number** vì đây là một con số dùng để so sánh (>, <); **Grade** (xếp loại học lực, A/B/C) có kiểu **Character** vì mỗi giá trị chỉ gồm đúng một ký tự.

(!) Lỗi thường gặp: The primary key must be chosen so it can *never* repeat and *never* be left blank across the whole life of the table. `StudentID` works because the school assigns a brand-new code to every student. A field such as `FirstName` or `LastName` must never be used alone as a primary key: two different students could easily share the same first name (or the same surname), which would instantly break the "always unique" rule, even if no such clash happens to exist in today's five rows.

9.1.1 Field data types

Every field in a table is restricted to one data type, chosen when the table is designed. Choosing the right data type matters because it controls what values can be stored, how much space they take up, and what operations (sorting, calculating, comparing) are possible on that field.

Data type	What it stores — example
Text / String	Any sequence of characters (letters, digits, symbols) that will not be used in calculations, e.g. `FirstName` = "Amelia", or an address.
Number / Integer	A whole number used for counting or comparing, e.g. `YearGroup` = 10, or a quantity in stock.
Real / Decimal	A number that may include a fractional (decimal) part, e.g. a student's average test score of 78.5, or a height in metres.
Date	A calendar date stored in a fixed format so it can be sorted and compared correctly, e.g. a student's date of birth, 14/03/2010.
Boolean / Logical	Exactly one of two values only: `True/False` (or Yes/No), e.g. whether a library book `IsOnLoan`.
Currency	A monetary amount, stored with a fixed number of decimal places and usually a currency symbol, e.g. the price of a book, $12.99.

Common field data types used when designing a database table.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mỗi trường trong bảng chỉ được gán **một kiểu dữ liệu** duy nhất khi thiết kế bảng. **Text/String** lưu chuỗi ký tự bất kỳ (không dùng để tính toán); **Number/Integer** lưu số nguyên dùng để đếm hoặc so sánh; **Real/Decimal** lưu số có phần thập phân; **Date** lưu ngày tháng theo định dạng cố định để có thể sắp xếp đúng theo thời gian; **Boolean/Logical** chỉ nhận đúng một trong hai giá trị (Đúng/Sai); **Currency** lưu số tiền với số chữ số thập phân cố định. Chọn đúng kiểu dữ liệu giúp cơ sở dữ liệu kiểm soát chặt chẽ dữ liệu nhập vào và cho phép các phép toán (so sánh, sắp xếp, tính toán) hoạt động chính xác.

9.1.2 A second sample table: Books

To practise SQL beyond a single table, this chapter also uses a small library catalogue. Each record describes one title held by a school library.

Sample table: Books

BookID	Title	Author	Genre	Year Published	Copies Available
B01	The Silent Orbit	Maria Chen	Science Fiction	2014	3
B02	Autumn Letters	James Whitfield	Fiction	2018	5
B03	The Data Detective	Priya Anand	Non-Fiction	2020	2
B04	Shadows of Kyoto	Haruto Sato	Fiction	2011	0
B05	Codebreakers	Elena Petrov	Non-Fiction	2016	4
B06	The Last Orchard	Maria Chen	Fiction	2019	1

Field data types: BookID = Text (primary key), Title = Text, Author = Text, Genre = Text, YearPublished = Number, CopiesAvailable = Number.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng **Books** là bảng dữ liệu mẫu thứ hai, mô phỏng danh mục sách của thư viện trường học, dùng để luyện thêm các câu truy vấn SQL đa dạng hơn ở phần sau của chương. `BookID` là khoá chính (duy nhất cho mỗi cuốn sách); `YearPublished` và `CopiesAvailable` có kiểu **Number** vì cả hai đều được dùng để so sánh (ví dụ: năm xuất bản sau 2015, hay số bản còn lại lớn hơn 0).

9.2 Why use a database rather than separate files

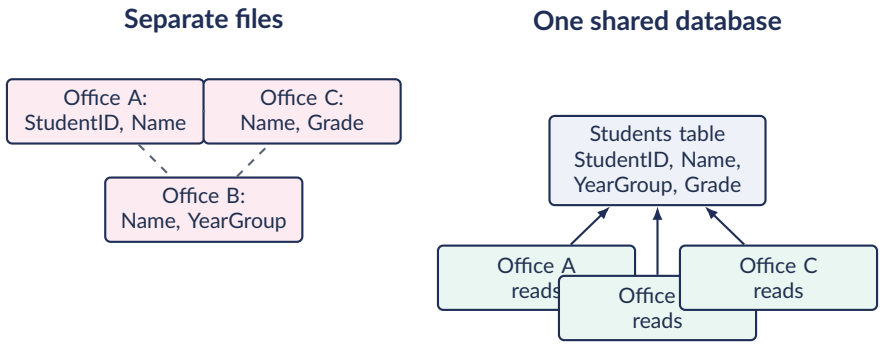
Before relational databases became standard, organisations often kept related information scattered across many separate, unlinked files (for example, a spreadsheet per department, or a paper card index per office). A single well-designed database avoids the problems this causes.

Consider the school in this chapter's examples. Suppose the front office keeps a spreadsheet of `StudentID`, `FirstName` and `LastName` for correspondence; the timetabling office keeps a completely separate spreadsheet of `FirstName`, `LastName` and `YearGroup` for room allocation; and the exams office keeps a third file of `FirstName`, `LastName` and `Grade` for report cards. If a student legally changes their surname, all three files must be found and edited individually — and if even one is missed, the school now holds two different surnames for the same student depending on which office is asked. A single `Students` table, read (not copied) by all three offices, removes this risk entirely: the surname is corrected once, and every office immediately sees the same, current value.

Khái niệm cốt lõi

Advantages of storing data in one shared database rather than in many separate files:

- **Reduced data duplication (redundancy):** the same piece of information (e.g. a student's surname) is stored once, in one table, rather than being copied into every file that happens to need it.
- **Easier, more consistent updates:** because a fact is stored in only one place, updating it (e.g. a change of surname) only has to be made once; with scattered files, every separate copy would have to be found and updated, and it is easy to miss one, leaving the data inconsistent.
- **Centralised control and security:** a database can be managed from one place, with access rights, backups and validation rules applied consistently to all the data, rather than each separate file being looked after (or forgotten about) independently.



Left: the same student's name is copied into three unlinked files — an update to one copy can leave the others out of date. Right: one shared table is updated once and read by every office.

VN **GIẢI THÍCH TIẾNG VIỆT**

Trước khi có cơ sở dữ liệu tập trung, cùng một thông tin (ví dụ tên học sinh) thường bị sao chép rải rác ở nhiều tệp tách biệt, không liên kết với nhau — mỗi phòng ban giữ một bản riêng. Điều này gây ra ba vấn đề chính mà một cơ sở dữ liệu duy nhất giải quyết được: **giảm trùng lặp dữ liệu** (chỉ lưu một bản duy nhất); **cập nhật dễ dàng và nhất quán hơn** (chỉ cần sửa một chỗ, tránh tình trạng các bản sao bị lệch nhau do quên cập nhật); và **kiểm soát & bảo mật tập trung** (quyền truy cập, sao lưu, quy tắc kiểm tra hợp lệ được áp dụng đồng bộ cho toàn bộ dữ liệu thay vì mỗi tệp riêng lẻ tự quản lý theo cách khác nhau).

9.3 Validation in databases

Chapter 7 introduced five general validation checks used to reject clearly wrong data before it is accepted: range, length, type, presence and format checks. These same checks are applied directly to the fields of a database table whenever a new record is entered, so that the table never ends up storing an impossible value.

Khái niệm cốt lõi

Applying validation checks to database fields:

- **Presence check** — rejects a record if a required field (especially the primary key) has been left blank. `StudentID` must never be empty, since the database relies on it to identify the record.
- **Range check** — rejects a value that falls outside the values that make sense for that field. `YearGroup` must be 10 or 11 at this school, so any other number is rejected; a range check can also apply to letters in order, e.g. `Grade` must fall between 'A' and 'C'.
- **Length check** — rejects a value with the wrong number of characters. Every `StudentID` must be exactly 4 characters long (the letter S followed by three digits).
- **Type check** — rejects a value that is not of the field's declared data type. `YearGroup` must contain a Number; text such as "Ten" is rejected.
- **Format check** — rejects a value that does not match a required pattern. Every `StudentID` must start with the letter S followed by exactly three digits, e.g. S001; a value beginning with a digit is rejected.

Validation only checks that a value is *plausible*; it cannot guarantee the value is *correct* (a genuine student mistakenly given `YearGroup` = 11 instead of 10 would still pass every check above).

VN **GIẢI THÍCH TIẾNG VIỆT**

Chương 7 đã giới thiệu năm loại kiểm tra hợp lệ (validation): kiểm tra phạm vi (range), độ dài (length), kiểu dữ liệu (type), sự hiện diện (presence) và định dạng (format). Khi áp dụng vào cơ sở dữ liệu, các kiểm tra này được gắn trực tiếp vào từng trường của bảng: `StudentID` không được để trống (presence) và phải đúng độ dài 4 ký tự (length) theo đúng khuôn mẫu "S" + 3 chữ số (format); `YearGroup` phải là kiểu Number (type) và chỉ được nhận giá trị 10 hoặc 11 (range). Cần lưu ý: validation chỉ đảm bảo dữ liệu **hợp lý**, chứ không đảm bảo dữ liệu đó **đúng sự thật** — một học sinh thực chất học lớp 10 nhưng bị nhập nhầm thành 11 vẫn sẽ vượt qua mọi kiểm tra trên.

Ví dụ mẫu · Worked example

A school secretary tries to add five candidate new records to the `Students` table. For each, state which validation check (if any) it fails.

Field & value	Check failed	Reason
StudentID = (blank)	Presence check	The primary key can never be left empty.
StudentID = "S6"	Length check	Must be exactly 4 characters (S + 3 digits); this is only 2.
StudentID = "12S4"	Format check	Must start with the letter S followed by three digits; this starts with a digit.
YearGroup = "Ten"	Type check	The field must hold a Number, not text.
YearGroup = 13	Range check	Only Year 10 and Year 11 students exist at this school.

A record such as StudentID = "S006", FirstName = "Ethan", LastName = "Brooks", YearGroup = 11, Grade = "B" passes every check above and can be added to the table.

Ví dụ mẫu · Worked example — validating a new Books record

A librarian proposes adding a new record to the `Books` table: `BookID` = "B7", `Title` = "The Glass Bridge", `Author` = "Omar Farid", `Genre` = "Fiction", `YearPublished` = 2027, `CopiesAvailable` = -1. Identify every field that fails validation.

Field & value	Check failed	Reason
BookID = "B7"	Length check	Every `BookID` must be exactly 3 characters (the letter B followed by two digits, e.g. B07); "B7" is only 2 characters.
YearPublished = 2027	Range check	Today's date is in 2026, so a book cannot already have been published in a future year; the valid range is up to the current year.
CopiesAvailable = -1	Range check	The number of copies physically available can never be negative; the valid range is 0 or more.

`Title`, `Author` and `Genre` are all plausible, non-blank text values and pass every check; only `BookID`, `YearPublished` and `CopiesAvailable` would be rejected as they stand.

(!) Lỗi thường gặp: Passing every validation check does *not* prove a record is correct — it only proves the data is a *plausible* value for its field. A range check on `YearGroup` accepts any value from 10 to 11; it cannot detect that a particular student's year group was typed in wrong if the wrong value (10 instead of 11, say) still lies inside that valid range. Confirming that entered data genuinely matches the real world requires **verification** (e.g. double-entry, or proofreading against the original source), which is a separate process from validation.

9.4 SQL SELECT queries

Once data is safely stored in a table, it needs to be searched, filtered and sorted. **SQL (Structured Query Language)** is the standard language used to do this. The single most important SQL statement for retrieving data is `SELECT`.

Khái niệm cốt lõi

The general structure of a simple SQL query is:

```
SELECT <field list>
FROM <table name>
WHERE <condition>
ORDER BY <field> ASC/DESC
```

`SELECT *` may be used instead of naming individual fields, to return *every* field of the matching records. The `WHERE` clause is optional (omitting it returns every record in the table); the `ORDER BY` clause is also optional (omitting it leaves the returned rows in no guaranteed particular order).

Clause	Purpose
`SELECT <fields>`	Chooses which field(s) of each matching record are displayed in the result. `SELECT *` displays all fields.
`FROM <table>`	States which table the records are read from.
`WHERE <condition>`	Filters the records: only rows for which the condition is true are included in the result.
`ORDER BY <field> ASC/DESC`	Sorts the returned rows by the stated field, ascending (`ASC`, the default) or descending (`DESC`).

The structure of an SQL `SELECT` query: each clause and what it controls.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cấu trúc chung của một câu truy vấn SQL: `SELECT` chọn những trường nào sẽ hiển thị (`SELECT *` nghĩa là hiển thị tất cả các trường); `FROM` cho biết đọc dữ liệu từ bảng nào; `WHERE` lọc ra chỉ những bản ghi thoả điều kiện; `ORDER BY` sắp xếp kết quả trả về theo một trường, tăng dần (`ASC`) hoặc giảm dần (`DESC`). Mệnh đề `WHERE` và `ORDER BY` đều không bắt buộc: bỏ `WHERE` sẽ trả về toàn bộ bản ghi; bỏ `ORDER BY` thì thứ tự các dòng trả về không được đảm bảo.

(!) Lỗi thường gặp: The four clauses of a `SELECT` query must always appear in the fixed order `SELECT...FROM...WHERE...ORDER BY` — writing `ORDER BY` before `WHERE`, for instance, is a syntax error, not merely a style choice. Two further common mistakes: forgetting the quotation marks around a text value in a condition (`WHERE Grade = A` instead of `WHERE Grade = 'A'`), and forgetting that `ORDER BY` sorts ascending (`ASC`) by default if `ASC/DESC` is left out entirely, which is easy to mistake for "no particular order".

Ví dụ mẫu · Worked example

Write an SQL query to display the first and last names of all Year 10 students, sorted alphabetically by surname.

```
SELECT FirstName, LastName
FROM Students
WHERE YearGroup = 10
ORDER BY LastName ASC
```

Tracing the query against the Students table: the `WHERE` clause keeps only the Year 10 rows — Amelia Turner, Chloe Nguyen and Isabella Novak. Sorting these three by `LastName` ascending (alphabetically) gives Nguyen, Novak, Turner. The final result, in order, is:

FirstName	LastName
Chloe	Nguyen
Isabella	Novak
Amelia	Turner

Comparison operators usable inside a `WHERE` condition: = (equal to), <> (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to). Text values in a condition are written inside quotation marks, e.g. `WHERE Grade = 'A'`; numeric values are written without quotation marks, e.g. `WHERE YearGroup = 10`.

Ví dụ mẫu · Worked example — <= and >=

The school librarian wants a low-stock report: every book with at most 1 copy currently available, together with the number of copies, ordered with the scarcest book first.

```
SELECT Title, CopiesAvailable
FROM Books
WHERE CopiesAvailable <= 1
ORDER BY CopiesAvailable ASC
```

Tracing: the `CopiesAvailable` values are 3, 5, 2, 0, 4, 1 for the six books in order. Only two satisfy `<= 1`: Shadows of Kyoto (0 copies) and The Last Orchard (1 copy). Sorting these ascending by `CopiesAvailable` keeps 0 before 1, so the report reads:

Title	CopiesAvailable
Shadows of Kyoto	0
The Last Orchard	1

Note that `CopiesAvailable < 1` would have returned only Shadows of Kyoto (0 copies), silently missing The Last Orchard; the <= operator is needed whenever the boundary value itself (here, exactly 1 copy) must be included in the result.

9.4.1 Combining conditions with AND / OR

A single `WHERE` clause is often not precise enough. Two (or more) conditions can be combined using the logical keywords `AND` and `OR`.

Khái niệm cốt lõi

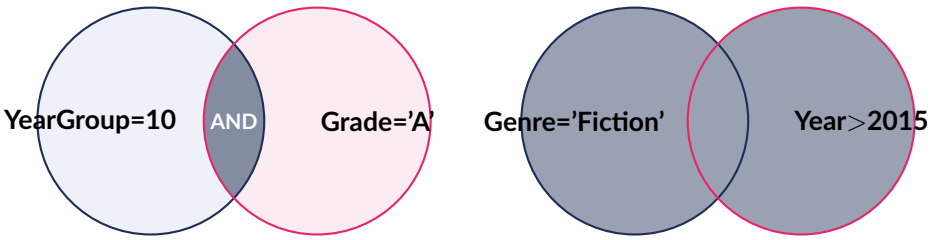
AND — a record is included in the result only if *every* condition joined by `AND` is true for that record (the conditions narrow the result down).

OR — a record is included in the result if *at least one* condition joined by `OR` is true for that record (the conditions widen the result: rows matching either side are kept, and any row matching both is still only listed once).

VN **GIẢI THÍCH TIẾNG VIỆT**

`AND` chỉ giữ lại những bản ghi thoả **đồng thời tất cả** các điều kiện — kết quả sẽ **thu hẹp lại**. `OR` giữ lại bản ghi chỉ cần thoả **ít nhất một** trong các điều kiện — kết quả sẽ **mở rộng ra**. Một lỗi rất phổ biến là nhầm lẫn hai từ khoá này: dùng `OR` khi ý muốn thực chất là "phải thoả cả hai"

sẽ trả về nhiều bản ghi hơn cần thiết, và ngược lại.



AND keeps only the dark overlap (both true). **OR** keeps everything in either circle.

AND keeps only records satisfying both conditions at once (the overlap); OR keeps records satisfying either condition (the whole shaded union).

Ví dụ mẫu · Worked example — `AND` on Students

Write an SQL query to list the ID and surname of Year 10 students who achieved Grade A.

```
SELECT StudentID, LastName
FROM Students
WHERE YearGroup = 10 AND Grade = 'A'
```

Tracing: Year 10 students are Amelia Turner (Grade A), Chloe Nguyen (Grade B) and Isabella Novak (Grade C). Only Amelia Turner satisfies *both* conditions at once. Result: one row — `S001, Turner`.

Ví dụ mẫu · Worked example — `OR` on Books

Write an SQL query to list the titles of books that are either Fiction, or were published after 2015 (or both).

```
SELECT Title
FROM Books
WHERE Genre = 'Fiction' OR YearPublished > 2015
```

Tracing, book by book:

Title	Genre='Fiction'?	Year>2015?	Included?
The Silent Orbit (2014, Sci-Fi)	No	No	No
Autumn Letters (2018, Fiction)	Yes	Yes	Yes
The Data Detective (2020, Non-Fic.)	No	Yes	Yes
Shadows of Kyoto (2011, Fiction)	Yes	No	Yes
Codebreakers (2016, Non-Fic.)	No	Yes	Yes
The Last Orchard (2019, Fiction)	Yes	Yes	Yes

Since `OR` needs only one side true, five of the six books are returned; only *The Silent Orbit* (Science Fiction and published in 2014) fails both conditions and is excluded.

(!) Lỗi thường gặp: `WHERE Genre = 'Fiction' OR YearPublished > 2015` does **not** mean "Fiction books published after 2015 only". That narrower meaning needs `AND`: `WHERE Genre = 'Fiction' AND YearPublished > 2015`. Always check, sentence by sentence, whether the

question really means "both must be true" (`AND`) or "either is enough" (`OR`) before writing the query.

Ví dụ mẫu · Worked example — `SELECT`, `WHERE...AND`, and `ORDER BY` together

Write an SQL query to list the title and year of publication of Fiction books published after 2015, with the earliest of these listed first.

```
SELECT Title, YearPublished
FROM Books
WHERE Genre = 'Fiction' AND YearPublished > 2015
ORDER BY YearPublished ASC
```

Tracing: the Fiction books are Autumn Letters (2018), Shadows of Kyoto (2011) and The Last Orchard (2019). Applying `AND YearPublished > 2015` removes Shadows of Kyoto (2011 is not after 2015), leaving Autumn Letters and The Last Orchard. Sorting these two by `YearPublished` ascending, 2018 comes before 2019, so the result is:

Title	YearPublished
Autumn Letters	2018
The Last Orchard	2019

This example shows all four clauses working together: `SELECT` chooses the two fields to display, `FROM` names the table, `WHERE...AND` filters using two conditions at once, and `ORDER BY...ASC` sorts what remains.

9.4.2 Wildcards with LIKE

Sometimes only *part* of a text value is known — for example, all surnames beginning with a certain letter. The `LIKE` keyword, combined with the `%` wildcard, searches for a pattern rather than an exact match.

Khái niệm cốt lõi

`LIKE` is used inside a `WHERE` clause instead of = to match a text pattern. The `%` symbol stands for *any sequence of zero or more characters*:

- `'N%'` — matches any value that **starts with** N.
- `'%Chen'` — matches any value that **ends with** Chen.
- `'%or%'` — matches any value that contains "or" **anywhere** within it.

VN **GIẢI THÍCH TIẾNG VIỆT**

`LIKE` dùng để so khớp một **mẫu (pattern)** thay vì so khớp chính xác từng ký tự như =. Ký hiệu đại diện `%` thay thế cho **bất kỳ chuỗi ký tự nào (kể cả rỗng)**: đặt `%` ở cuối (`'N%'`) để tìm giá trị **bắt đầu bằng** N; đặt `%` ở đầu (`'%Chen'`) để tìm giá trị **kết thúc bằng** Chen; đặt `%` ở cả hai đầu (`'%or%'`) để tìm giá trị **chứa** chuỗi đó ở bất kỳ vị trí nào.

Ví dụ mẫu · Worked example

Write a query to list all students whose surname begins with the letter N.

```
SELECT *
```

```
FROM Students
WHERE LastName LIKE 'N%'
```

Tracing: the surnames in the table are Turner, Ng, Nguyen, Osei, Novak. Three begin with N: **Ng**, **Nguyen** and **Novak**. Result: the full records for Brandon Ng (S002), Chloe Nguyen (S003) and Isabella Novak (S005) — three rows in total.

Ví dụ mẫu · Worked example

Predict the result of the following query on the Books table.

```
SELECT Title
FROM Books
WHERE Author LIKE '%Chen'
```

Tracing: the `Author` field contains Maria Chen (B01), James Whitfield (B02), Priya Anand (B03), Haruto Sato (B04), Elena Petrov (B05), and Maria Chen again (B06). The pattern `'%Chen'` matches any author name *ending* in "Chen": this matches B01 and B06 only. Result: **The Silent Orbit** and **The Last Orchard**.

Ví dụ mẫu · Worked example

Predict the result of the following query, given that matching is not case-sensitive.

```
SELECT Title
FROM Books
WHERE Title LIKE '%or%'
```

Tracing, title by title: The Silent Orbit (contains "Or" in *Orbit* ⇒ match), Autumn Letters (no "or"), The Data Detective (no "or"), Shadows of Kyoto (no "or"), Codebreakers (no "or"), The Last Orchard (contains "Or" in *Orchard* ⇒ match). Result: **The Silent Orbit** and **The Last Orchard**.

9.4.3 Forms and reports

In practice, users almost never type raw SQL into a live production database. Instead, a database is usually accessed indirectly through a purpose-built, on-screen **form** — a user-friendly window with labelled boxes and buttons that guides data entry (and can apply validation automatically as each field is completed). For the `Students` table, a form might present labelled boxes for `StudentID`, `FirstName`, `LastName`, a drop-down list restricted to 10 or 11 for `YearGroup`, and a drop-down list restricted to A, B or C for `Grade` — so that the office worker filling it in can never accidentally type an out-of-range value in the first place, rather than relying only on validation to reject it afterwards. Once data has been queried, the results are often presented as a formatted **report** — a clearly laid-out, print-ready summary (with headings, totals and groupings) designed for a human to read, rather than the raw table of rows an SQL query returns. A report generated from the `Books` table might, for instance, group titles by `Genre`, print a heading above each group, and add a subtotal of `CopiesAvailable` at the end of each group — none of which an SQL `SELECT` statement produces directly on its own.

VN **GIẢI THÍCH TIẾNG VIỆT**

Trong thực tế, người dùng hiếm khi gõ trực tiếp câu lệnh SQL vào cơ sở dữ liệu sản xuất. Thay vào đó, dữ liệu thường được nhập qua một **biểu mẫu (form)** thân thiện trên màn hình, với các

ô nhập và nhãn rõ ràng, giúp áp dụng kiểm tra hợp lệ ngay khi nhập. Sau khi truy vấn, kết quả thường được trình bày dưới dạng **báo cáo (report)** — một bản trình bày đã định dạng đẹp, có tiêu đề, tổng hợp và nhóm dữ liệu, dễ đọc và có thể in ra, thay vì chỉ hiển thị bảng dữ liệu thô như kết quả trực tiếp của một câu lệnh SQL.

9.5 Practice questions

Bài tập luyện tập

Which of the following best defines a *primary key*?

(A) The first field listed in a table

(B) A field guaranteed to hold a unique, non-blank value for every record in the table

(C) Any field whose data type is Number

(D) A field used only for sorting a table

Lời giải chi tiết

(B). A primary key's defining property is that its value can never repeat and can never be left blank across the whole table, so it always identifies exactly one record. Position in the table (A), data type (C) and sorting use (D) are all irrelevant to whether a field qualifies as a primary key.

Bài tập luyện tập

Explain why `FirstName` alone would be a poor choice of primary key for the `Students` table, even though no two first names currently happen to clash in the five sample rows.

Lời giải chi tiết

A primary key must be guaranteed unique for *every* record, now and in the future, not merely unique by coincidence in the data that currently exists. Two different students could easily be given the same first name (for example, two students both called "Daniel" could join the school next year); if `FirstName` were the primary key, the database would then be unable to tell the two records apart, breaking the guarantee a primary key must provide. `StudentID` is a safer choice because the school deliberately assigns it as a unique code to each student.

Bài tập luyện tập

Which data type would be most appropriate for a field called `DateOfBirth`?

(A) Text

(B) Number

(C) Date

(D) Boolean

Lời giải chi tiết

(C) Date. Storing a date of birth as the Date type (rather than as Text) allows the database to validate that it is a genuine calendar date, to sort records correctly by age, and to compare or calculate with dates (e.g. working out someone's current age).

Bài tập luyện tập

Predict the exact result of the following query, run against the `Students` table.

```
SELECT FirstName, Grade
FROM Students
WHERE Grade = 'A'
```

Lời giải chi tiết

Scanning the Grade field: Amelia Turner = A, Brandon Ng = B, Chloe Nguyen = B, Daniel Osei = A, Isabella Novak = C. Two records have Grade = A: Amelia and Daniel. Result:

FirstName	Grade
Amelia	A
Daniel	A

Bài tập luyện tập

Write an SQL query to display every field of all Year 11 students, sorted by first name in descending order.

Lời giải chi tiết

```
SELECT *
FROM Students
WHERE YearGroup = 11
ORDER BY FirstName DESC
```

The Year 11 students are Brandon Ng and Daniel Osei. Sorted by `FirstName` descending (Z to A), Daniel comes before Brandon, so the result lists Daniel Osei's full record first, then Brandon Ng's.

Bài tập luyện tập

Write an SQL query to display the full records of all students whose surname begins with the letter N.

Lời giải chi tiết

```
SELECT *
FROM Students
WHERE LastName LIKE 'N%'
```

The surnames Ng, Nguyen and Novak all begin with N, so this returns three full records: Brandon Ng, Chloe Nguyen and Isabella Novak. (Turner and Osei do not begin with N and are excluded.)

Bài tập luyện tập

Which of these is **not** one of the five standard validation checks?

(A) Range check
(B) Format check
(C) Popularity check
(D) Presence check

Lời giải chi tiết

(C) Popularity check. The five standard validation checks are range, length, type, presence and format. "Popularity" is not a recognised validation check — it is not possible to validate how popular a value is; validation only checks whether a value is plausible for its field.

Bài tập luyện tập

A new record is proposed for the Students table with `StudentID` left blank. Which validation check does this fail, and why is this check especially important for a primary key field?

Lời giải chi tiết

This fails a **presence check**. A presence check rejects any record where a required field has been left empty. It is especially critical for a primary key because the whole purpose of a primary key is to uniquely identify every record; a blank primary key value would give the database no way to distinguish that record from any other blank-keyed record, and no way to look the record up reliably.

Bài tập luyện tập

A new record is proposed with `YearGroup = 13`. Which validation check does this fail?

(A) Presence check
(B) Range check
(C) Length check
(D) Type check

Lời giải chi tiết

(B) Range check. The school only has Year 10 and Year 11 students, so the valid range for `YearGroup` is 10 to 11 inclusive. A range check rejects any value, such as 13, that falls outside this permitted range, even though 13 is a perfectly valid *number* (so a type check would not catch it).

Bài tập luyện tập

Explain, using an example, the difference between *validation* and *verification* as applied to a new Students record.

Lời giải chi tiết

Validation is an automatic check, performed by the software, that rejects data which is implausible for its field — for example, rejecting `YearGroup = 13` because it lies outside the valid range 10–11. **Verification** is a check that the data entered genuinely matches the real-world source it was copied from — for example, a second person re-typing the same student's details and the system comparing the two entries, or the secretary reading the record back to the student to confirm it is correct. A record can pass every validation check yet still be factually wrong (e.g. a genuine Year 10 student mistakenly entered as `YearGroup = 11`, which is still a valid value); only verification would be able to catch that kind of error.

Bài tập luyện tập

Predict the exact result of the following query, run against the `Books` table.

```
SELECT Title, Author
FROM Books
WHERE Genre = 'Fiction'
ORDER BY YearPublished ASC
```

Lời giải chi tiết

The Fiction books are Autumn Letters (2018), Shadows of Kyoto (2011) and The Last Orchard (2019). Sorting by `YearPublished` ascending gives 2011, 2018, 2019. Result:

Title	Author
Shadows of Kyoto	Haruto Sato
Autumn Letters	James Whitfield
The Last Orchard	Maria Chen

Bài tập luyện tập

Write an SQL query to list the titles of books that currently have at least one copy available **and** were published before 2015.

Lời giải chi tiết

```
SELECT Title
FROM Books
WHERE CopiesAvailable > 0 AND YearPublished < 2015
```

Checking each book against *both* conditions: only *The Silent Orbit* (3 copies available, published 2014) satisfies both at once. *Shadows of Kyoto* was published in 2011 (before 2015) but has 0 copies available, so it fails the first condition and is correctly excluded. Result: **The Silent Orbit** only.

Bài tập luyện tập

Predict how many rows are returned by the following query, and name the one book (if any) that is excluded.

```
SELECT *
FROM Books
WHERE Genre = 'Fiction' OR YearPublished > 2015
```

Lời giải chi tiết

Because `OR` only requires one side to be true, every book is included *except* one that is both non-Fiction *and* published in 2015 or earlier. Checking all six: only *The Silent Orbit* (Science Fiction, 2014) is neither Fiction nor published after 2015, so it is the sole book excluded. The query returns **5 rows**: Autumn Letters, The Data Detective, Shadows of Kyoto, Codebreakers, The Last Orchard.

Bài tập luyện tập

Which SQL keyword is used to sort the rows returned by a query?

(A) WHERE
(B) FROM
(C) ORDER BY
(D) SELECT

Lời giải chi tiết

(C) ORDER BY. `SELECT` chooses which fields to display, `FROM` states the source table, and `WHERE` filters which rows are included — none of these three control the order of the rows in the result; only `ORDER BY`, together with `ASC` or `DESC`, does.

Bài tập luyện tập

Write an SQL query to list the ID and last name of Year 11 students who achieved Grade C.

Lời giải chi tiết

```
SELECT StudentID, LastName
FROM Students
WHERE YearGroup = 11 AND Grade = 'C'
```

Tracing this against the table: the Year 11 students are Brandon Ng (Grade B) and Daniel Osei (Grade A); neither has Grade C. So this query correctly returns **no rows at all** — an empty result is a perfectly valid, correct answer when no record satisfies every condition.

Bài tập luyện tập

Predict the exact result of the following query.

```
SELECT StudentID, LastName
FROM Students
WHERE YearGroup = 10 AND Grade <> 'C'
ORDER BY LastName DESC
```

Lời giải chi tiết

The Year 10 students are Amelia Turner (A), Chloe Nguyen (B) and Isabella Novak (C). Excluding Grade = 'C' removes Isabella Novak, leaving Amelia Turner and Chloe Nguyen. Sorting the two remaining surnames, Turner and Nguyen, in descending (Z to A) order places Turner before Nguyen. Result:

StudentID	LastName
S001	Turner
S003	Nguyen

Bài tập luyện tập

Write an SQL query to display all students who are either in Year 11, or who achieved Grade C (or both).

Lời giải chi tiết

```
SELECT *
FROM Students
WHERE YearGroup = 11 OR Grade = 'C'
```

Year 11 students: Brandon Ng, Daniel Osei. Grade C students: Isabella Novak. Since `OR` keeps any row matching either condition, the result is the full records of **three** students: Brandon Ng, Daniel Osei and Isabella Novak.

Bài tập luyện tập

Which of these correctly matches book titles that contain the substring "Data" anywhere within the title?

(A) Title LIKE 'Data'
(B) Title LIKE 'Data%'
(C) Title LIKE '%Data%'
(D) Title = '%Data%'

Lời giải chi tiết

(C). Placing % on *both* sides of the search text matches the substring anywhere within the field — at the start, middle, or end. Option (A) matches only the exact word "Data" with nothing else at all; option (B) matches only titles that *start* with "Data"; option (D) uses = instead of `LIKE`, so the wildcard % would be treated as a literal character rather than a pattern, and would not match anything in this table.

Bài tập luyện tập

Write an SQL query to list the title and year published of all books published in 2016 or later, sorted with the most recent book first.

Lời giải chi tiết

```
SELECT Title, YearPublished
FROM Books
WHERE YearPublished >= 2016
ORDER BY YearPublished DESC
```

Books with `YearPublished >= 2016`: Autumn Letters (2018), The Data Detective (2020), Codebreakers (2016), The Last Orchard (2019). Sorted by year descending (most recent first):

Title	YearPublished
The Data Detective	2020
The Last Orchard	2019
Autumn Letters	2018
Codebreakers	2016

Bài tập luyện tập

A librarian wants to add a new record to the Books table: `BookID` = "B07", `Title` = "Night Trains", `Author` = "Sofia Reyes", `Genre` = "Fiction", `YearPublished` = "Twenty Twenty-Two", `CopiesAvailable` = 2. Identify the one field that would fail validation, and name the check.

Lời giải chi tiết

`YearPublished` = "Twenty Twenty-Two" fails a **type check**: the field is defined as Number, but text has been entered instead of a numeric year (e.g. 2022). Every other field is a plausible value for its data type: `BookID` is a non-blank text code, `Genre` is text, and `CopiesAvailable` is a whole number.

Bài tập luyện tập

Which statement correctly distinguishes a *form* from a *report* in a database system?

(A) A form is used to enter and edit data on screen; a report is a formatted, print-ready presentation of retrieved data

(B) A form and a report are two names for the same screen

(C) A report is used only to delete records

(D) A form always contains SQL code visible to the user

Lời giải chi tiết

(A). A form is a user-friendly on-screen interface for entering or editing individual records (often with built-in validation), while a report takes the results of a query and lays them out in a clear, formatted, print-ready way for a person to read, typically including headings, groupings or totals rather than a raw table of rows.

Bài tập luyện tập

Write an SQL query to display the ID, title and genre of all books, sorted alphabetically by title.

Lời giải chi tiết

```
SELECT BookID, Title, Genre
FROM Books
ORDER BY Title ASC
```

No `WHERE` clause is needed since all six books should appear. Sorted alphabetically by title: Autumn Letters, Codebreakers, Shadows of Kyoto, The Data Detective, The Last Orchard, The Silent Orbit.

Bài tập luyện tập

Explain one advantage of storing student and book information in a single shared database rather than in several separate, unlinked spreadsheet files kept by different staff members.

Lời giải chi tiết

Any one clearly explained advantage is acceptable, for example: **reduced duplication** — each fact (such as a student's surname, or a book's title) is stored only once in the shared database, instead of being copied into every separate spreadsheet that happens to need it. This also makes **updates more consistent**: changing the fact once in the shared table automatically makes the change available to everyone, whereas with separate files each copy would need to be found and updated individually, risking some copies being missed and becoming out of date. A shared database additionally allows **centralised security and access control**, applied consistently to all the data rather than left to each member of staff individually.

Bài tập luyện tập

Predict the exact result of the following query.

```
SELECT Title
FROM Books
WHERE CopiesAvailable = 0
```

Lời giải chi tiết

Scanning `CopiesAvailable`: The Silent Orbit = 3, Autumn Letters = 5, The Data Detective = 2, Shadows of Kyoto = 0, Codebreakers = 4, The Last Orchard = 1. Only *Shadows of Kyoto* has exactly 0 copies available. Result: a single row, **Shadows of Kyoto**.

Bài tập luyện tập

A field called `IsOnLoan` is added to the Books table to record whether each copy is currently borrowed. Which data type is most appropriate, and why?

(A) Text, because "on loan" is a phrase

(B) Boolean, because the field can only ever be one of two states: true or false

(C) Currency, because loans may involve a fine

(D) Date, because loans have a due date

Lời giải chi tiết

(B) Boolean. The field only ever needs to represent exactly one of two states — on loan, or not on loan — which is precisely what a Boolean (True/False) field is designed to store. Text (A) would allow inconsistent entries such as "yes", "Y" or "on loan" for the same meaning; Currency (C) and Date (D) describe different pieces of information (a fine amount, or a due date) that could exist as separate fields, but neither represents the on-loan status itself.

Bài tập luyện tập

Write an SQL query to list the ID and surname of all students who are *not* in Year 10.

Lời giải chi tiết

```
SELECT StudentID, LastName
FROM Students
WHERE YearGroup <> 10
```

The <> operator means "not equal to". Every student who is not Year 10 is Year 11, so this returns Brandon Ng (S002) and Daniel Osei (S004).

Bài tập luyện tập

A candidate SQL query is written as: `SELECT Title FROM Books WHERE Genre = Fiction`. Explain the error in this query and correct it.

Lời giải chi tiết

The error is that the text value `Fiction` has been written without quotation marks. In SQL, text (string) values inside a `WHERE` condition must be enclosed in quotation marks so they are recognised as literal text rather than, for example, a field name; without quotes, most database systems would reject the query or fail to find any matching `Genre` field. The corrected query is:

```
SELECT Title FROM Books WHERE Genre = 'Fiction'
```

Bài tập luyện tập

Predict the exact result of the following query, and explain what would change if `AND` were replaced with `OR`.

```
SELECT Title
FROM Books
WHERE Genre = 'Non-Fiction' AND CopiesAvailable > 2
```

Lời giải chi tiết

The Non-Fiction books are The Data Detective (2 copies) and Codebreakers (4 copies). Only Codebreakers has `CopiesAvailable > 2`, so with `AND` the query returns just **Codebreakers**. If `AND` were replaced with `OR`, a book would be included whenever *either* condition alone is true: every Non-Fiction book (The Data Detective and Codebreakers) *plus* every book with more than 2 copies available regardless of genre (The Silent Orbit with 3, Autumn Letters with 5, and Codebreakers again with 4) — giving a much larger result of four distinct titles: The Silent Orbit, Autumn Letters, The Data Detective, and Codebreakers.

Bài tập luyện tập

State two fields, other than `StudentID`, that could sensibly be combined to form a primary key for a table if no single field is unique on its own, and explain why combining them can work even when neither field is unique alone.

Lời giải chi tiết

For example, `FirstName` combined with `DateOfBirth` could be used as a combined (composite) primary key. Although two students could share the same first name, and separately two students could share the same date of birth, it is far less likely that two students share *both* the same first name *and* the same date of birth at once. A composite key is unique because it is the *combination* of values across the chosen fields that must be unique for every record, not each field individually — so fields that are not unique alone can still form a valid, unique key together. (In practice, schools still prefer a dedicated code such as `StudentID`, since even a composite key like this could theoretically still clash.)

Bài tập luyện tập

What is the role of a *Database Management System (DBMS)*?

(A) It is the physical device used to back up a database

(B) It is the software that creates, stores and lets users query and update a database, enforcing its structure and validation rules

(C) It is another name for a single table within a database

(D) It is a printed report generated from a query

Lời giải chi tiết

(B). A DBMS is the software layer that sits between the stored data and every program or user that needs it: it creates and maintains the table structure, applies validation rules to incoming data, and processes queries (such as SQL `SELECT` statements) on behalf of whatever is requesting the data. It is not a physical device (A), not a single table (C), and not the printed output of a query (D) — it is the system that manages all of these.

Bài tập luyện tập

Write an SQL query to list the ID, first name and surname of every student whose surname contains the letter combination "ov" anywhere within it, sorted alphabetically by first name.

Lời giải chi tiết

```
SELECT StudentID, FirstName, LastName
FROM Students
WHERE LastName LIKE '%ov%'
ORDER BY FirstName ASC
```

Checking each surname for the substring "ov": Turner (no), Ng (no), Nguyen (no), Osei (no), Novak (contains "ov" in Nov**a**k — yes). Only Isabella Novak (S005) matches, so `ORDER BY` has no further effect on a single-row result. The query returns one row: `S005, Isabella, Novak`.

Boolean Logic

Mục tiêu Unit: Cổng logic cơ bản (AND, OR, NOT), cổng NAND/NOR/XOR, bảng chân trị (truth table), mạch logic kết hợp nhiều cổng, suy ra biểu thức Boolean từ mạch và vẽ mạch từ biểu thức, các định luật đại số Boolean cơ bản (De Morgan), và mạch cộng bán phần (half adder).

10.1 Basic logic gates and truth tables

A **logic gate** is a digital circuit that outputs a single Boolean value (`1`/`TRUE` or `0`/`FALSE`) based on the values of its inputs. Every gate has one or more inputs and exactly one output. A **truth table** lists, for every possible combination of input values, the resulting output — it is the complete, unambiguous definition of what a gate (or a whole circuit) does.

Khái niệm cốt lõi

The three most basic gates are:

- **AND** — two inputs; output is 1 only when *both* inputs are 1.
- **OR** — two inputs; output is 1 when *at least one* input is 1.
- **NOT** — exactly *one* input; the output is simply the input inverted (0 becomes 1, 1 becomes 0). NOT is the only gate with a single input.

A	B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

NOT: $\overline{0} = 1,\ \overline{1} = 0.$

Figure 10.1: AND, OR and NOT gates, with their standard truth tables.

AND outputs 1 only when *both* inputs are 1. **OR** outputs 1 when *at least one* input is 1. **NOT** has a single input and inverts it (also drawn as a triangle with a small circle, or "bubble", on its output).

VN **GIẢI THÍCH TIẾNG VIỆT**

AND: kết quả 1 chỉ khi **cả hai** đầu vào là 1. **OR**: kết quả 1 khi **ít nhất một** đầu vào là 1. **NOT**: chỉ có 1 đầu vào, đảo ngược giá trị (0 thành 1, 1 thành 0). Ký hiệu $\overline{A}$ nghĩa là "NOT A".

Ví dụ mẫu · Worked example

For $A = 1$ and $B = 0$, evaluate the output of an AND gate and an OR gate.
AND: both inputs must be 1 for the output to be 1; here $B = 0$, so the output is **0**.
OR: at least one input must be 1; here $A = 1$, so the output is **1**.

(!) Lỗi thường gặp: AND and OR are easy to swap by accident under exam pressure. A quick memory check: AND is the *stricter* gate (needs *everything* to be true, like the English word "and"); OR is the *more lenient* gate (needs only *one* thing to be true). If in doubt, always re-derive the answer directly from the truth table rather than relying on memory alone.

10.2 NAND, NOR and XOR

Beyond AND, OR and NOT, three further gates are required for Cambridge IGCSE: NAND, NOR and XOR. Each can be understood either as a truth table in its own right, or as a simple combination of the basic gates already met.

Khái niệm cốt lõi

NAND ("NOT AND") = NOT(AND) — outputs 1 in every case *except* when both inputs are 1, where it outputs 0.

NOR ("NOT OR") = NOT(OR) — outputs 1 only when *both* inputs are 0; any other combination gives 0.

XOR ("exclusive OR") outputs 1 only when the two inputs are *different* from each other (exactly one input is 1); it outputs 0 when the inputs are the *same* (both 0 or both 1).

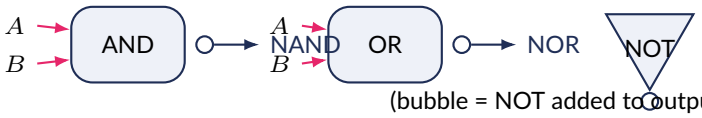


Figure 10.2: standard gate-symbol convention — a small circle ("bubble") on a gate's output line always means the whole gate's result is inverted; this is exactly how NAND is drawn as an AND shape with a bubble, and NOR as an OR shape with a bubble.

A	B	AND	OR	NAND	NOR	XOR
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Figure 10.3: combined 7-column truth table for AND, OR, NAND, NOR and XOR.

VN **GIẢI THÍCH TIẾNG VIỆT**

NAND = đảo của AND (chỉ 0 khi cả hai đầu vào là 1, các trường hợp còn lại đều là 1). **NOR** = đảo của OR (chỉ 1 khi cả hai đầu vào là 0). **XOR**: kết quả 1 khi hai đầu vào **khác nhau** (một cái 0, một cái 1); kết quả 0 khi hai đầu vào **giống nhau**. Trên sơ đồ mạch, một vòng tròn nhỏ (bubble) ở đầu ra của một cổng luôn có nghĩa là kết quả của cổng đó bị đảo (NOT) — đây là lý do NAND được vẽ giống hệt AND nhưng có thêm bubble, và NOR được vẽ giống OR nhưng có thêm bubble.

Ví dụ mẫu · Worked example

Using the truth table in Figure 10.3, state the output of a NOR gate and of an XOR gate when $A = 0, B = 1$.

Reading the row where $A = 0, B = 1$: NOR $= 0$ (since at least one input is 1, OR would be 1, so NOT OR is 0); XOR $= 1$ (since the two inputs, 0 and 1, are different).

(!) Lỗi thường gặp: Do not confuse NOR with NAND. A useful check: NOR is 1 in exactly *one* row (00), while NAND is 1 in exactly *three* rows (00, 01, 10) — NAND is 0 only in the single row where both inputs are 1. If your truth table for NAND has more than one row equal to 0, it has been built incorrectly.

10.2.1 NAND and NOR as two-gate circuits

Although NAND and NOR are treated as single, standard gates, it is worth seeing explicitly that each one can be built from two of the more basic gates already met — this is exactly what the "bubble" notation in Figure 10.2 represents on a physical circuit diagram.

Figure 10.2b: NAND built as an AND gate followed by a NOT gate (left); NOR built as an OR gate followed by a NOT gate (right) — confirming NAND = NOT(AND) and NOR = NOT(OR) directly from wiring, not just from the definition.

VN **GIẢI THÍCH TIẾNG VIỆT**

Hình trên minh hoạ trực tiếp bằng dây nối rằng **NAND** chính là một cổng AND nối tiếp với một cổng NOT, và **NOR** chính là một cổng OR nối tiếp với một cổng NOT — không chỉ là định nghĩa bằng lời mà còn là cấu trúc mạch thật sự, đúng như quy ước hình vẽ "bubble" (vòng tròn nhỏ ở đầu ra) đã trình bày ở Hình 10.2.

10.2.2 Operator precedence in Boolean expressions

Just as arithmetic expressions follow an order of operations (brackets, then multiplication/division, then addition/subtraction), Boolean expressions follow a fixed order too, which matters whenever an expression is written *without* full brackets around every operation.

Khái niệm cốt lõi

Unless brackets say otherwise, Boolean operators are evaluated in this order: **(1) brackets first**, **(2) NOT**, **(3) AND**, **(4) OR** (NAND/NOR/XOR are normally always shown with explicit brackets around their inputs in Cambridge-style questions, precisely to avoid any ambiguity). In this book, every expression is written with full brackets specifically so that this ordering never needs to be relied upon — but recognising it is useful for interpreting an expression that a question presents without brackets.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cũng như biểu thức số học có thứ tự thực hiện phép tính, biểu thức Boolean cũng có thứ tự cố định khi không có dấu ngoặc: **ngoặc trước**, rồi đến **NOT**, rồi **AND**, cuối cùng mới đến **OR**. Trong sách này, mọi biểu thức đều được viết kèm đầy đủ dấu ngoặc để không bao giờ cần dựa vào quy tắc thứ tự này, nhưng vẫn cần biết quy tắc để đọc hiểu một biểu thức được cho mà không có ngoặc.

Ví dụ mẫu · Worked example

Without adding any brackets, evaluate $Y =$ NOT A AND B OR C for $A = 1, B = 0, C = 1$, applying the standard precedence (NOT, then AND, then OR).

Step 1 (NOT first): NOT $A =$ NOT $1 = 0$.
Step 2 (AND next): (NOT A) AND $B = 0$ AND $0 = 0$.
Step 3 (OR last): 0 OR $C = 0$ OR $1 = 1$.
So $Y = 1$. Written fully bracketed, this expression is $Y =$ ((NOT A) AND B) OR C — exactly the same circuit structure as Figure 10.4, but with the inputs relabelled.

10.3 Combining gates into logic circuits

Complex logic circuits are built by wiring several gates together, so that the output of one gate becomes an input to the next. To find the truth table of an entire circuit, do *not* try to evaluate the whole expression in one step — work through the circuit gate by gate, in the order the signal actually flows, computing one sub-expression's column fully before moving on to the next.

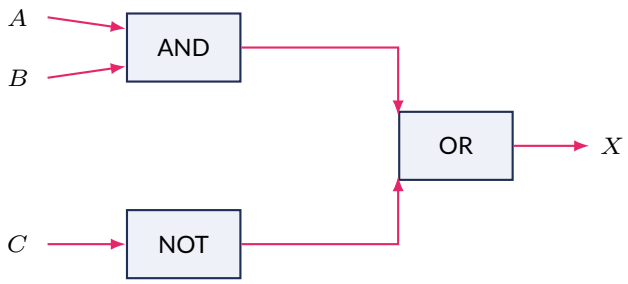


Figure 10.4: circuit for $X = (A$ AND $B)$ OR $(\text{NOT } C)$.

Ví dụ mẫu · Worked example

Construct the full truth table for $X = (A$ AND $B)$ OR $(\text{NOT } C)$.

A	B	C	A AND B	NOT C	X
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Work column by column: compute A AND B, then NOT C, then OR the two results together for each row.

VN **GIẢI THÍCH TIẾNG VIỆT**

Để lập bảng chân trị cho một **mạch logic kết hợp nhiều cổng**, hãy tính **từng cổng con một**, theo đúng thứ tự tín hiệu đi qua mạch (ví dụ tính A AND B và NOT C trước, rồi mới OR hai kết quả lại), thay vì cố gắng tính trực tiếp cả biểu thức cùng lúc.

10.3.1 Worked example: a more complex four-input circuit

Circuits with more inputs and more gates are handled with exactly the same column-by-column method — the only difference is that there are more sub-expressions to compute before reaching

the final output, and more rows to fill in (four independent binary inputs give $2^4 = 16$ possible combinations).

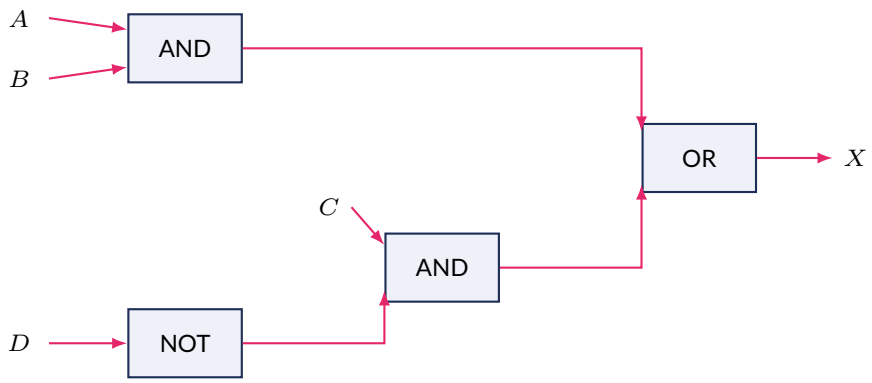


Figure 10.5: circuit for $X = (A$ AND $B)$ OR $(C$ AND $($NOT $D))$, with four independent inputs.

Ví dụ mẫu · Worked example

Construct the truth table for $X = (A$ AND $B)$ OR $(C$ AND $($NOT $D))$. Work sub-expression by sub-expression: first $P = A$ AND B, then $Q =$ NOT D, then $R = C$ AND Q, and finally $X = P$ OR R.

A	B	C	D	P=AANDB	Q=$\overline{D}$	R=CANDQ	X=PORR
0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	0	0	0
0	1	0	0	0	1	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	1	1	1
0	1	1	1	0	0	0	0
1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	1	1	1
1	0	1	1	0	0	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	0	0	1
1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	1

Notice that $X = 1$ whenever $P = 1$ (i.e. whenever $A = B = 1$, regardless of C and D — see the bottom four rows), *or* whenever $R = 1$ (i.e. whenever $C = 1$ and $D = 0$, regardless of A and B — see rows 3, 7, 11, 15). This matches the circuit exactly: X is 1 if *either* branch feeding the final OR gate is 1.

VN **GIẢI THÍCH TIẾNG VIỆT**

Với mạch có **4 đầu vào**, bảng chân trị sẽ có $2^4 = 16$ dòng (thay vì 8 dòng như mạch 3 đầu vào). Cách làm không đổi: tính từng **biểu thức con** theo đúng thứ tự cổng (ở đây là $P = A$ AND B, rồi $Q = \overline{D}$, rồi $R = C$ AND Q, cuối cùng $X = P$ OR R), thêm mỗi biểu thức con là một cột riêng trong bảng để tránh nhầm lẫn và dễ kiểm tra lại kết quả.

10.4 Deriving expressions from circuits, and drawing circuits from expressions

Cambridge IGCSE exams frequently ask for the reverse of what has been practised so far: reading a Boolean expression *off* a given circuit diagram, or drawing a circuit diagram *from* a given Boolean

expression. Both skills use the same underlying idea — follow the wires from input to output (or from expression to gates) one gate at a time.

Khái niệm cốt lõi

To derive an expression from a circuit: start at the inputs and label the output of each gate, in signal order, with the sub-expression it computes, using the previous gate's label as the new input where wires connect; the label on the final gate's output is the complete Boolean expression for the circuit.

To draw a circuit from an expression: identify the *last* operation performed (this becomes the final gate, whose output is the circuit's output); work backwards, adding one gate for every AND/OR/NOT/NAND/NOR/XOR in the expression, and connect each gate's inputs to either an original input variable or the output of another gate, exactly as the brackets in the expression dictate.

VN **GIẢI THÍCH TIẾNG VIỆT**

Suy ra biểu thức từ mạch: bắt đầu từ các đầu vào, đặt nhãn cho đầu ra của từng cổng theo đúng thứ tự tín hiệu, dùng nhãn của cổng trước làm đầu vào cho cổng sau; nhãn ở cổng cuối cùng chính là biểu thức Boolean đầy đủ của mạch. **Vẽ mạch từ biểu thức**: xác định phép toán được thực hiện **sau cùng** (đây sẽ là cổng cuối, cho ra đầu ra của mạch), rồi vẽ ngược lại từng cổng ứng với từng AND/OR/NOT/NAND/NOR/XOR có trong biểu thức, nối đầu vào của mỗi cổng theo đúng cấu trúc dấu ngoặc của biểu thức.

10.4.1 Worked example: deriving an expression from a circuit

Ví dụ mẫu · Worked example

Deriving an expression from a circuit. A circuit has three inputs A, B, C. A and B feed into a NOR gate; the output of that NOR gate and input C feed into an AND gate, whose output is Y. Write the Boolean expression for Y.

Label the NOR gate's output first, since it is computed before the final gate: NOR gate output $= \overline{A \text{ OR } B}$ (NOR of A and B). This label, together with C, feeds the AND gate, so:

$$Y = (\overline{A \text{ OR } B}) \text{ AND } C$$

Checking with a quick sample: if $A = 0, B = 0, C = 1$, then $\overline{A \text{ OR } B} = \overline{0} = 1$, so $Y = 1$ AND $1 = 1$, which is consistent with both inputs to the NOR gate being 0 (giving NOR $= 1$) and $C = 1$.

10.4.2 Worked example: drawing a circuit from an expression

Ví dụ mẫu · Worked example

Drawing a circuit from an expression. Draw a circuit for $Z = (A \text{ XOR } B) \text{ AND } (\overline{C})$.

The *last* operation performed is the outer AND, so the final gate is an AND gate with two inputs: the result of A XOR B, and the result of $\overline{C}$ (NOT C). Working backwards: draw an XOR gate fed by inputs A and B; draw a separate NOT gate fed by input C; connect the outputs of *both* of these gates into the two inputs of a final AND gate, whose output is Z.

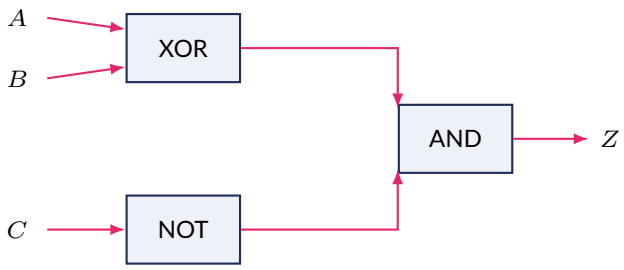


Figure 10.6: circuit drawn from the expression $Z = (A$ XOR $B)$ AND $(\overline{C})$.

(!) Lỗi thường gặp: When drawing a circuit from an expression, always identify the *outermost* (last-performed) operation first and make that the *final* gate. A common mistake is to draw the gates in the order the operators are written left to right, which can connect the wrong wires together when the expression contains brackets.

10.5 Basic Boolean algebra laws

Just as ordinary algebra has laws (e.g. $a + b = b + a$), Boolean algebra has laws that let an expression be rewritten into an equivalent form without changing its truth table. Two are required here: the **double negation law** and **De Morgan's laws**.

Khái niệm cốt lõi

Double negation law: applying NOT twice returns the original value:

$$\overline{\overline{A}} = A$$

De Morgan's laws: inverting an AND/OR expression as a whole is the same as inverting each input separately and *swapping* AND for OR (or OR for AND):

$$\overline{A \text{ AND } B} = \overline{A} \text{ OR } \overline{B} \qquad \overline{A \text{ OR } B} = \overline{A} \text{ AND } \overline{B}$$

VN **GIẢI THÍCH TIẾNG VIỆT**

Định luật phủ định kép (double negation): NOT hai lần thì quay lại giá trị ban đầu, $\overline{\overline{A}} = A$. **Định luật De Morgan**: đảo (NOT) toàn bộ một biểu thức AND/OR thì bằng với việc đảo **từng đầu vào riêng lẻ** rồi **đổi AND thành OR** (hoặc OR thành AND). Đây là công cụ hữu ích để viết lại (rewrite) một biểu thức Boolean sang dạng khác nhưng vẫn cho cùng kết quả.

Ví dụ mẫu · Worked example

Verify $\overline{A \text{ AND } B} = \overline{A} \text{ OR } \overline{B}$ using a truth table.

A	B	A AND B	$\overline{A \text{ AND } B}$	$\overline{A}$	$\overline{B}$	$\overline{A}$ OR $\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Figure 10.7: truth-table verification of De Morgan's first law.

The two final columns, $\overline{A \text{ AND } B}$ and $\overline{A}$ OR $\overline{B}$, are identical in every row $(1, 1, 1, 0)$ — confirming the two expressions are logically equivalent, so either may be used interchangeably.

Ví dụ mẫu · Worked example

Verify $\overline{A \text{ OR } B} = \overline{A} \text{ AND } \overline{B}$ (De Morgan's second law) using a truth table.

A	B	A OR B	$\overline{A \text{ OR } B}$	$\overline{A}$	$\overline{B}$	$\overline{A}$ AND $\overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Again, the last two columns match exactly in every row $(1, 0, 0, 0)$ — this row of results is also exactly the NOR truth table, confirming that both $\overline{A}$ OR $\overline{B}$ and $\overline{A}$ AND $\overline{B}$ describe a NOR gate.

10.5.1 Worked example: applying De Morgan's law to rewrite an expression

Ví dụ mẫu · Worked example

Rewrite $Y = \overline{(A \text{ AND } B) \text{ OR } C}$ using De Morgan's law so that the outer NOT is removed, then verify the two forms agree using $A = 1, B = 0, C = 0$.

Let $P = A$ AND B. Then $Y = \overline{P \text{ OR } C}$. By De Morgan's second law, $\overline{P \text{ OR } C} = \overline{P}$ AND $\overline{C}$, and $\overline{P} = \overline{A \text{ AND } B}$, which by De Morgan's first law equals $\overline{A}$ OR $\overline{B}$. So:

$$Y = (\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{C}$$

Check at $A = 1, B = 0, C = 0$**:** original form: A AND $B = 1$ AND $0 = 0$; 0 OR $C = 0$ OR $0 = 0$; $Y = \overline{0} = 1$. Rewritten form: $\overline{A} = 0, \overline{B} = 1$, so $\overline{A}$ OR $\overline{B} = 0$ OR $1 = 1$; $\overline{C} = 1$; $Y = 1$ AND $1 = 1$. Both forms agree: $Y = 1$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ví dụ trên áp dụng De Morgan **hai lần liên tiếp** để loại bỏ dấu NOT bao ngoài toàn bộ biểu thức: trước tiên đảo OR thành AND (đổi dấu từng phần bên trong), sau đó đảo tiếp phần AND ở bên trong bằng định luật De Morgan còn lại. Luôn kiểm tra lại bằng cách thay số cụ thể vào cả hai dạng biểu thức — nếu kết quả khớp nhau ở mọi bộ giá trị thử, hai biểu thức chắc chắn tương đương.

10.6 The half adder circuit

Chapter 1 showed that adding two binary digits can produce a result requiring two digits: $1 + 1 = 10$ in binary (a 0 in that column, with a 1 **carried** into the next column to the left). A **half adder** is the fundamental combinational logic circuit that performs exactly this operation on a single pair of bits.

Khái niệm cốt lõi

A **half adder** takes two single binary inputs, A and B, and produces two outputs:

- **Sum** — the units-column result of adding A and B: Sum $= A$ XOR B.
- **Carry** — whether the addition overflows into the next column: Carry $= A$ AND B.

It is called a "half" adder because it only adds two bits with no carry-in from a previous column; a **full adder** (which also accepts a carry-in) is built by combining two half adders, but is not required at this level.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mạch cộng bán phần (half adder) là mạch tổ hợp cơ bản dùng để cộng **hai bit nhị phân đơn lẻ** A và B, cho ra hai đầu ra: **Sum** (chữ số kết quả ở hàng đơn vị) $=$ A XOR B, và **Carry** (bit nhớ sang cột kế tiếp) $=$ A AND B. Sở dĩ gọi là "bán phần" (half) vì mạch này chỉ cộng hai bit mà **không nhận bit nhớ từ cột trước** — muốn cộng có bit nhớ vào cần ghép hai mạch half adder lại thành **mạch cộng đầy đủ (full adder)**, nội dung này nằm ngoài chương trình IGCSE.

The link to binary addition: when $A = 1$ and $B = 1$, ordinary binary addition gives $1+1 = 10_2$ — a Sum digit of **0** with a Carry of **1** into the next column. This is exactly what Sum $=$ A XOR B and Carry $=$ A AND B produce: XOR gives 0 when both inputs are the same (both 1), and AND gives 1 only when both inputs are 1.

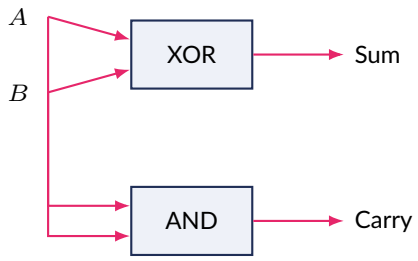


Figure 10.8: half adder circuit — one XOR gate produces Sum, one AND gate produces Carry, both fed by the same two inputs A and B.

A	B	Sum (A XOR B)	Carry (A AND B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 10.9: full truth table for the half adder.

Ví dụ mẫu · Worked example

Trace the half adder for $A = 1, B = 1$ and explain the result in terms of binary addition.

Sum $=$ A XOR B $=$ 1 XOR 1 $=$ 0 (the inputs are the same, so XOR gives 0). Carry $=$ A AND B $=$ 1 AND 1 $=$ 1 (both inputs are 1, so AND gives 1). Reading Carry and Sum together gives 10_2, which is exactly $1 + 1$ in binary — the half adder has correctly computed that adding two 1s produces a 0 in this column with a 1 carried to the next column.

(!) Lỗi thường gặp: It is easy to swap Sum and Carry by mistake. Remember: **Carry** can only ever be 1 in the single row where *both* inputs are 1 (exactly like AND); **Sum** is 1 in the two rows where the inputs *differ* (exactly like XOR). If your Carry column has more than one row equal to 1, it has been built using the wrong gate.

10.7 Identifying a gate from its truth table

Examination questions sometimes work in reverse: instead of asking for a truth table given a gate, they give the truth table (or describe circuit behaviour in words) and ask which single standard gate it matches. This is solved by comparing the given pattern of 0s and 1s directly against the six reference patterns already met.

Reference summary of all six standard gates, in the input order $(A, B) = (0,0), (0,1), (1,0), (1,1)$:

Gate	Output pattern	Rule
AND	$0, 0, 0, 1$	1 only if both inputs are 1
OR	$0, 1, 1, 1$	1 if at least one input is 1
NAND	$1, 1, 1, 0$	0 only if both inputs are 1 (opposite of AND)
NOR	$1, 0, 0, 0$	1 only if both inputs are 0 (opposite of OR)
XOR	$0, 1, 1, 0$	1 only if the two inputs differ

Every one of these five two-input patterns is different from every other, so any truth table with exactly two inputs must match *at most one* of them (or none, if it does not correspond to any of the five). NOT is the only gate with a single input, with pattern $\overline{0} = 1, \overline{1} = 0$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng trên tổng hợp lại **5 mẫu kết quả** (output pattern) của các cổng hai đầu vào theo đúng thứ tự $(A, B) = (0,0), (0,1), (1,0), (1,1)$. Vì cả 5 mẫu này đều **khác nhau hoàn toàn**, nên khi đề bài cho một bảng chân trị bất kỳ, chỉ cần so khớp cột kết quả với 5 mẫu này là xác định được ngay đó là cổng nào (hoặc không khớp cổng chuẩn nào cả).

Ví dụ mẫu · Worked example

A circuit with two inputs A, B has been tested and produces the following outputs in order: for $(A, B) = (0,0)$, output $= 1$; for $(0,1)$, output $= 0$; for $(1,0)$, output $= 0$; for $(1,1)$, output $= 0$. Identify which single standard gate this circuit is equivalent to.

The observed pattern, in the standard input order, is $1, 0, 0, 0$. Comparing against the reference summary above, this exactly matches **NOR** (1 only when both inputs are 0). No other listed gate produces this pattern, so the circuit must be a NOR gate (or something logically equivalent to one, such as an OR gate followed by a NOT gate).

Ví dụ mẫu · Worked example

A word problem states: "a warning light turns on whenever exactly one of two sensors, A and B, detects a fault, but stays off if neither or both sensors detect a fault." Identify the single gate this describes, and write its Boolean expression.

"Exactly one of two inputs is 1" (light on when the sensors *differ*, off when they are the *same*) is precisely the definition of **XOR**. The Boolean expression for the warning light L is $L = A$ XOR B.

10.8 Practice

Bài tập luyện tập

For inputs $A = 1$, $B = 0$, what is the output of a NAND gate?

(A) 0

(B) 1

(C) Cannot be determined

(D) Depends on C

Lời giải chi tiết

NAND = NOT(AND). A AND $B = 1$ AND $0 = 0$. NOT $0 = 1$. Output = $\boxed{1}$. **(B).**

Bài tập luyện tập

Which single gate outputs 1 only when its two inputs are *different* from each other?

(A) AND
(B) NOR
(C) XOR
(D) NAND

Lời giải chi tiết

XOR (exclusive OR) outputs 1 only when exactly one input is 1 (the inputs differ); it outputs 0 when both inputs are the same (00 or 11). **(C).**

Bài tập luyện tập

Complete the truth table for Y = NOT(A OR B) for all four combinations of A and B, and state which single gate this is equivalent to.

Lời giải chi tiết

A OR B: $0, 1, 1, 1$ for $(A, B) = (0,0), (0,1), (1,0), (1,1)$. Applying NOT: $Y = 1, 0, 0, 0$ respectively. This truth table $(1, 0, 0, 0)$ is exactly the definition of **NOR** — so Y = NOT(A OR B) is equivalent to a single **NOR gate**.

Bài tập luyện tập

A security system arms an alarm only if *both* the front-door switch (A) and back-door switch (B) are closed, **unless** an override key switch (C) is on, which always disarms it regardless of A and B. Write a Boolean expression for "alarm armed" (X), and evaluate X when $A = 1, B = 1, C = 1$.

Lời giải chi tiết

"Both doors closed" is A AND B; the override should force the alarm off whenever $C = 1$, so we need NOT C combined with AND: $X = (A$ AND $B)$ AND (NOT C). Evaluate at $A = 1, B = 1, C = 1$: A AND $B = 1$; NOT C = NOT $1 = 0$; $X = 1$ AND $0 = 0$ — the alarm is **not** armed, because the override key is on, exactly as required.

Bài tập luyện tập

Simplify the circuit description: two inputs A and B first both pass through separate NOT gates, and the two outputs are then combined with an AND gate. Write the resulting Boolean expression for the output Z, and identify which single standard gate produces the *same* truth table as Z.

(A) $Z = \overline{A}$ AND $\overline{B}$, equivalent to NOR
(B) $Z = \overline{A}$ OR $\overline{B}$, equivalent to NAND
(C) $Z = A$ AND B, equivalent to AND
(D) $Z = \overline{A}$ AND $\overline{B}$, equivalent to XOR

Lời giải chi tiết

$Z = \overline{A}$ AND $\overline{B}$. Testing all four input combinations gives $1, 0, 0, 0$ for $(A, B) = (0,0), (0,1), (1,0), (1,1)$ — identical to the NOR truth table. **(A)**.

Bài tập luyện tập

State the output of a NOR gate for inputs $A = 0$, $B = 0$.

(A) 0
(B) 1
(C) Undefined
(D) Depends on the circuit

Lời giải chi tiết

NOR outputs 1 only when both inputs are 0. Here $A = 0, B = 0$, so NOR $= \boxed{1}$. **(B)**.

Bài tập luyện tập

Trace the circuit $X = (A$ AND $B)$ OR $(\text{NOT } C)$ (Figure 10.4) for $A = 0, B = 1, C = 1$, showing each sub-expression's value.

Lời giải chi tiết

A AND $B = 0$ AND $1 = 0$. NOT $C =$ NOT $1 = 0$. $X = 0$ OR $0 = 0$. This matches row 4 of the truth table in the worked example (A=0, B=1, C=1 ⇒ X=0).

Bài tập luyện tập

For the four-input circuit $X = (A$ AND $B)$ OR $(C$ AND $(\text{NOT } D))$ (Figure 10.5), evaluate X for $A = 0, B = 1, C = 1, D = 1$.

Lời giải chi tiết

$P = A$ AND $B = 0$ AND $1 = 0$. $Q = \overline{D} = \overline{1} = 0$. $R = C$ AND $Q = 1$ AND $0 = 0$. $X = P$ OR $R = 0$ OR $0 = 0$. This matches row 8 of the 16-row truth table (A=0, B=1, C=1, D=1 ⇒ X=0).

Bài tập luyện tập

In the same circuit $X = (A$ AND $B)$ OR $(C$ AND $(\text{NOT } D))$, evaluate X for $A = 1, B = 1, C = 0, D = 0$.

(A) 0
(B) 1
(C) Cannot be determined
(D) X is undefined when $C = 0$

Lời giải chi tiết

$P = A$ AND $B = 1$ AND $1 = 1$. Since $P = 1$, the OR gate output is automatically 1 regardless of the other branch (no need even to compute Q and R). $X = \boxed{1}$. **(B)**.

Bài tập luyện tập

A circuit has two inputs, A and B, feeding directly into a single NOR gate, whose output feeds into a NOT gate, producing output Y. Derive the Boolean expression for Y and simplify by naming the equivalent single gate for the whole circuit (use the double negation law).

Lời giải chi tiết

NOR gate output: $\overline{A \text{ OR } B}$. Passing this through a NOT gate: $Y = \overline{\overline{A \text{ OR } B}}$. By the double negation law, $\overline{\overline{X}} = X$ for any X, so $Y = A$ OR B. The whole two-gate circuit is therefore logically equivalent to a single **OR gate**.

Bài tập luyện tập

Draw (describe in words, gate by gate) the circuit that implements $W = \overline{A}$ OR $(B$ AND $C)$.

Lời giải chi tiết

The last operation performed is the outer OR, so the final gate is an OR gate with two inputs. Working backwards: one input to the final OR gate is $\overline{A}$, produced by a NOT gate fed directly by A. The other input is B AND C, produced by an AND gate fed by B and C. Connect the NOT gate's output and the AND gate's output into the two inputs of the final OR gate, whose output is W.

Bài tập luyện tập

A circuit has inputs A, B, C. A and B feed an XOR gate; the XOR gate's output and C feed a NAND gate, whose output is M. Write the Boolean expression for M, then evaluate M for $A = 1, B = 0, C = 1$.

Lời giải chi tiết

$M = \overline{(A \text{ XOR } B) \text{ AND } C}$ (NAND of the XOR output and C). Evaluate at $A = 1, B = 0, C = 1$: A XOR $B = 1$ XOR $0 = 1$ (inputs differ). $(A$ XOR $B)$ AND $C = 1$ AND $1 = 1$. $M = \overline{1} = 0$.

Bài tập luyện tập

State the double negation law and use it to simplify $\overline{\overline{\overline{A}}}$ (NOT applied three times to A).

(A) A

(B) $\overline{A}$

(C) 1

(D) 0

Lời giải chi tiết

Double negation: $\overline{\overline{A}} = A$. Applying it to the innermost pair of NOTs in $\overline{\overline{\overline{A}}}$ leaves a single remaining NOT: $\overline{\overline{\overline{A}}} = \overline{A}$. **(B)**.

Bài tập luyện tập

State De Morgan's first law (for $\overline{A \text{ AND } B}$) and use it to rewrite $\overline{A \text{ AND } B}$ without a NOT applied to the whole expression.

Lời giải chi tiết

De Morgan's first law: $\overline{A \text{ AND } B} = \overline{A} \text{ OR } \overline{B}$. Applying it directly: $\overline{A \text{ AND } B}$ is rewritten as $\overline{A}$ OR $\overline{B}$, where the NOT now applies only to each input separately, and AND has become OR.

Bài tập luyện tập

State De Morgan's second law (for $\overline{A \text{ OR } B}$), and verify it for the single case $A = 1, B = 0$.

Lời giải chi tiết

De Morgan's second law: $\overline{A \text{ OR } B} = \overline{A} \text{ AND } \overline{B}$. At $A = 1, B = 0$: left side: A OR $B = 1$ OR $0 = 1$, so $\overline{A \text{ OR } B} = \overline{1} = 0$. Right side: $\overline{A} = 0, \overline{B} = 1$, so $\overline{A}$ AND $\overline{B} = 0$ AND $1 = 0$. Both sides equal 0, confirming the law holds for this case.

Bài tập luyện tập

Using De Morgan's laws, rewrite $Y = \overline{(A \text{ OR } B) \text{ AND } C}$ so that the outer NOT no longer applies to the whole expression.

(A) $Y = (\overline{A} \text{ AND } \overline{B}) \text{ OR } \overline{C}$

(B) $Y = (\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{C}$

(C) $Y = (A \text{ AND } B) \text{ OR } C$

(D) $Y = (\overline{A} \text{ AND } \overline{B}) \text{ AND } \overline{C}$

Lời giải chi tiết

Let $P = A$ OR B, so $Y = \overline{P \text{ AND } C}$. By De Morgan's first law, $\overline{P \text{ AND } C} = \overline{P}$ OR $\overline{C}$. Now $\overline{P} = \overline{A \text{ OR } B}$, which by De Morgan's second law equals $\overline{A}$ AND $\overline{B}$. So $Y = (\overline{A} \text{ AND } \overline{B})$ OR $\overline{C}$. **(A)**.

Bài tập luyện tập

What are the two outputs of a half adder called, and what Boolean expression (in terms of inputs A and B) computes each one?

Lời giải chi tiết

The two outputs are **Sum** and **Carry**. Sum $= A$ XOR B (1 when the inputs differ). Carry $= A$ AND B (1 only when both inputs are 1).

Bài tập luyện tập

Explain, in terms of binary addition, why the half adder's Carry output is 1 only when $A = 1$ and $B = 1$.

Lời giải chi tiết

In binary, $1+1=10_2$: this produces a result that needs *two* digits, so there must be a carry of 1 into the next column, with 0 remaining in the current column (Sum). Every other combination ($0+0=0$, $0+1=1$, $1+0=1$) fits in a single binary digit with no overflow, so Carry is 0 in those three cases — exactly matching the AND gate, which is the only gate among those studied that is 1 in precisely one row (both inputs 1).

Bài tập luyện tập

Using the half adder truth table (Figure 10.9), state Sum and Carry for $A=0, B=1$.

(A) Sum $=0$, Carry $=0$

(B) Sum $=1$, Carry $=0$

(C) Sum $=0$, Carry $=1$

(D) Sum $=1$, Carry $=1$

Lời giải chi tiết

Sum $=A$ XOR $B=0$ XOR $1=1$ (inputs differ). Carry $=A$ AND $B=0$ AND $1=0$ (not both inputs are 1). **(B)**.

Bài tập luyện tập

Why is the circuit described in this chapter called a "half" adder rather than simply an "adder"?

Lời giải chi tiết

A half adder only adds two single bits, A and B, with no provision for a carry-*in* from a previous, less-significant column. A **full adder**, which accepts a carry-in as a third input (needed when adding multi-bit binary numbers, where every column after the first may receive a carry from the column before it), is built by combining two half adders together.

Bài tập luyện tập

A logic circuit has two inputs, A and B. Both feed a NAND gate, and both also feed a NOR gate; the outputs of the NAND gate and the NOR gate then feed an AND gate, whose output is F. Determine, by evaluating all four rows, whether F is equivalent to any of the standard single gates already studied.

Lời giải chi tiết

Build the full table. NAND: $1,1,1,0$ for $(A,B)=(00,01,10,11)$. NOR: $1,0,0,0$ for the same order. $F=$ NAND AND NOR: row $(0,0)$: 1 AND $1=1$; row $(0,1)$: 1 AND $0=0$; row $(1,0)$: 1 AND $0=0$; row $(1,1)$: 0 AND $0=0$. So $F=1,0,0,0$ — this is exactly the **NOR** truth table, so F is equivalent to a single NOR gate (in fact, NAND(A,B) AND NOR(A,B) $=$ NOR(A,B) for all inputs, since NAND is only 0 when NOR is already 0).

Bài tập luyện tập

Two lamps, L_1 and L_2, must both switch on together only when a light sensor (A, 1 = dark) detects darkness *and* a motion sensor (B, 1 = motion detected) detects motion, but never when a manual override switch (C, 1 = override on) is set, which always forces the lamps off. Write the Boolean expression for "lamps on" (L), and state whether $L=1$ when $A=1, B=1, C=0$.

Lời giải chi tiết

$L = (A \text{ AND } B) \text{ AND } (\overline{C})$. At $A = 1, B = 1, C = 0$: A AND $B = 1$; $\overline{C} = \overline{0} = 1$; $L = 1$ AND $1 = 1$ — yes, the lamps are on, since it is dark, motion is detected, and the override is not active.

Bài tập luyện tập

Which of the following correctly states De Morgan's law used to rewrite $\overline{A \text{ OR } B}$?

(A) $\overline{A \text{ OR } B} = \overline{A} \text{ OR } \overline{B}$

(B) $\overline{A \text{ OR } B} = A \text{ AND } B$

(C) $\overline{A \text{ OR } B} = \overline{A} \text{ AND } \overline{B}$

(D) $\overline{A \text{ OR } B} = \overline{A} \text{ AND } B$

Lời giải chi tiết

De Morgan's second law states $\overline{A \text{ OR } B} = \overline{A} \text{ AND } \overline{B}$: invert each input separately and swap OR for AND. **(C)**.

Bài tập luyện tập

Derive the Boolean expression for the circuit in Figure 10.6 ($Z = (A \text{ XOR } B) \text{ AND } (\overline{C})$) directly from the gate description (without looking at the expression), then evaluate Z for $A = 0, B = 0, C = 1$.

Lời giải chi tiết

The XOR gate (fed by A, B) and the NOT gate (fed by C) both feed into the final AND gate, giving $Z = (A \text{ XOR } B) \text{ AND } (\overline{C})$. At $A = 0, B = 0, C = 1$: A XOR $B = 0$ XOR $0 = 0$ (inputs are the same); $\overline{C} = \overline{1} = 0$; $Z = 0$ AND $0 = 0$.

Bài tập luyện tập

A four-input circuit computes $X = (A \text{ AND } B) \text{ OR } (C \text{ AND } (\text{NOT } D))$. For how many of the 16 possible input combinations is $X = 1$? (Use Figure 10.5's truth table.)

(A) 6

(B) 7

(C) 8

(D) 9

Lời giải chi tiết

Reading Figure 10.5's table row by row, $X = 1$ whenever $C{=}1, D{=}0$ (rows with A, B taking any of $00, 01, 10$, since the fourth such combination $A{=}B{=}1$ is already counted below) — namely rows $(0, 0, 1, 0)$, $(0, 1, 1, 0)$, $(1, 0, 1, 0)$ — and whenever $A{=}B{=}1$ regardless of C, D: $(1, 1, 0, 0)$, $(1, 1, 0, 1)$, $(1, 1, 1, 0)$, $(1, 1, 1, 1)$. That is $3 + 4 = \boxed{7}$ combinations out of 16. **(B)**.

Bài tập luyện tập

A circuit's output Y is defined as $Y = \overline{A} \text{ AND } \overline{B} \text{ AND } \overline{C}$ (three inputs, each inverted, then ANDed together). For how many of the 8 possible combinations of A, B, C is $Y = 1$?

(A) 0
(C) 2
(B) 1
(D) 4

Lời giải chi tiết

An AND of three terms is 1 only when *every* term is 1, so $Y = 1$ only when $\overline{A} = 1$, $\overline{B} = 1$ and $\overline{C} = 1$ simultaneously — that is, only when $A = 0, B = 0, C = 0$. This is exactly **1** combination out of the 8 possible. **(B)**.

Bài tập luyện tập

A circuit's output R is defined as $R = A$ OR B OR C (three inputs combined with OR). For how many of the 8 possible combinations of A, B, C is $R = 0$?

(A) 0
(C) 2
(B) 1
(D) 3

Lời giải chi tiết

An OR of several terms is 0 only when *every* term is 0, so $R = 0$ only when $A = 0, B = 0, C = 0$ — exactly **1** combination out of 8 (all other 7 combinations give $R = 1$, since at least one input is 1). **(B)**.

Bài tập luyện tập

Two half adders are available. Explain in one or two sentences why a single half adder is not sufficient to add two multi-bit binary numbers (e.g. two 4-bit numbers) column by column, and what extra input a full adder provides to solve this problem.

Lời giải chi tiết

When adding multi-bit binary numbers column by column, every column after the least significant one may receive a **carry-in** from the column to its right, in addition to its own two data bits — but a half adder only has two inputs (A and B) and cannot accept this extra carry-in. A **full adder** solves this by taking three inputs (the two data bits *and* a carry-in), producing its own Sum and Carry-out, so that full adders can be chained together, one per column, to add numbers of any length.

Bài tập luyện tập

Without adding brackets, evaluate $W = A$ OR B AND NOT C for $A = 0, B = 1, C = 1$, applying standard Boolean operator precedence (NOT, then AND, then OR).

Lời giải chi tiết

Precedence order: NOT first, then AND, then OR. NOT C = NOT $1 = 0$. B AND (NOT C) = 1 AND $0 = 0$. $W = A$ OR $0 = 0$ OR $0 = 0$. Fully bracketed, this expression is $W = A$ OR (B AND (NOT C)).

Bài tập luyện tập

Show, using a full truth table for both sides, that NAND built as "AND followed by NOT" (Figure 10.2b) really does produce the same output as the standard NAND column in Figure 10.3, for all four input combinations of A and B.

Lời giải chi tiết

A AND B: $0, 0, 0, 1$ for $(A, B) = (0, 0), (0, 1), (1, 0), (1, 1)$. Applying NOT to each of these: $1, 1, 1, 0$. This matches the standard NAND column from Figure 10.3 exactly $(1, 1, 1, 0)$ — confirming that "AND followed by NOT" is indeed a correct two-gate construction of NAND for every input combination.

Ôn tập nhanh: **Biểu diễn dữ liệu**: đổi qua lại nhị phân/thập phân/hex, số bù hai (two's complement) để biểu diễn số âm, phép dịch bit (shift), nén dữ liệu mất/không mất dữ liệu. **Truyền dữ liệu**: kiểm tra chẵn lẻ (parity check), tổng kiểm tra (checksum), giao thức yêu cầu lặp lại tự động (ARQ), chữ số kiểm tra (check digit). **Phần cứng**: chu trình **fetch–decode–execute**, vai trò các thanh ghi (PC, MAR, MDR, ACC, CIR); phân biệt **RAM** (khả biến, lưu chương trình/dữ liệu đang chạy), **ROM** (bất biến, chứa firmware/BIOS), và **cache** (rất nhanh, dung lượng nhỏ, nằm giữa CPU và RAM). **Phần mềm**: **compiler** dịch toàn bộ mã nguồn thành mã máy trước khi chạy, báo lỗi ở cuối; **interpreter** dịch và chạy từng dòng một, dừng ngay khi gặp lỗi. **Internet**: phân biệt **Internet** (hạ tầng mạng toàn cầu) và **World Wide Web** (dịch vụ trang web chạy trên Internet); vai trò của **DNS** (chuyển tên miền thành địa chỉ IP); phân biệt **phishing** (lừa qua email/tin nhắn giả mạo) và **pharming** (chuyển hướng người dùng đến trang web giả mạo bằng cách thay đổi DNS). **Tự động hoá và AI**: vòng lặp phản hồi của robot (cảm biến → bộ xử lý → bộ truyền động → cảm biến); các thành phần của hệ chuyên gia (expert system): cơ sở tri thức (knowledge base), động cơ suy luận (inference engine), giao diện người dùng. **Thuật toán**: **tìm kiếm nhị phân (binary search)** cần dữ liệu đã sắp xếp, **tìm kiếm tuần tự (linear search)** không cần; **sắp xếp nổi bọt (bubble sort)** và **sắp xếp chèn (insertion sort)**; các phép **kiểm tra hợp lệ (validation)**: range/length/type/presence/format/check digit, và phân biệt với **verification** (xác minh dữ liệu nhập đúng). **Lập trình**: các cấu trúc mã giả Cambridge (`IF...THEN...ELSE...ENDIF`, `FOR...NEXT`, `WHILE...ENDWHILE`, `REPEAT...UNTIL`); mảng (array) một chiều/hai chiều; thủ tục (procedure) và hàm (function); truyền tham số theo giá trị (by value) và theo tham chiếu (by reference). **Cơ sở dữ liệu**: câu lệnh SQL `SELECT ... FROM ... WHERE ... ORDER BY ...` để truy vấn, lọc và sắp xếp dữ liệu. **Logic Boolean**: cổng AND/OR/NOT/NAND/NOR/XOR và bảng chân trị của từng cổng; lập bảng chân trị cho mạch nhiều cổng theo đúng thứ tự tín hiệu; định luật De Morgan; mạch cộng bán phần (half adder) với Sum $=$ A XOR B và Carry $=$ A AND B.

Đồng hành cùng 8020 English

Cảm ơn bạn đã học cùng bộ giáo trình quốc tế 8020. **Điểm thật · Thầy thật · Kết quả thật** — nhiều học viên chỉ cần 1–2 khoá để đạt kết quả mong muốn.

Chương trình đào tạo tại 8020 English

IELTS Academic/General · **SAT** · PTE · Duolingo · TOEFL | **AP** (College Board) · **IB** Diploma · **Cambridge** IGCSE / A-Level | săn học bổng du học.

Vì sao chọn 8020 English?

- **100% giáo viên** đạt tối thiểu 8.0 IELTS hoặc 1500 SAT — đội ngũ Giáo sư · Tiến sĩ · Thạc sĩ tốt nghiệp trường top đầu thế giới (Carnegie Mellon — #1 Hoa Kỳ về Khoa học Máy tính).
- **Kèm 1–1** mọi độ tuổi; lớp sĩ số nhỏ 2–5 bạn (đa số dưới 10 bạn/lớp).
- Lộ trình cá nhân hoá, theo sát tiến độ từng học viên; lớp OFFLINE tại TP.HCM & ONLINE toàn quốc.

Bảng vàng học viên

AP 5/5 — Calculus AB/BC	**AP** 4–5 — Biology, Chemistry	**IB** 90/100 — Economics
SAT 1590 / 1570 / 1550	**IELTS** 8.0–9.0	**Duolingo** 110 · **PTE** 90

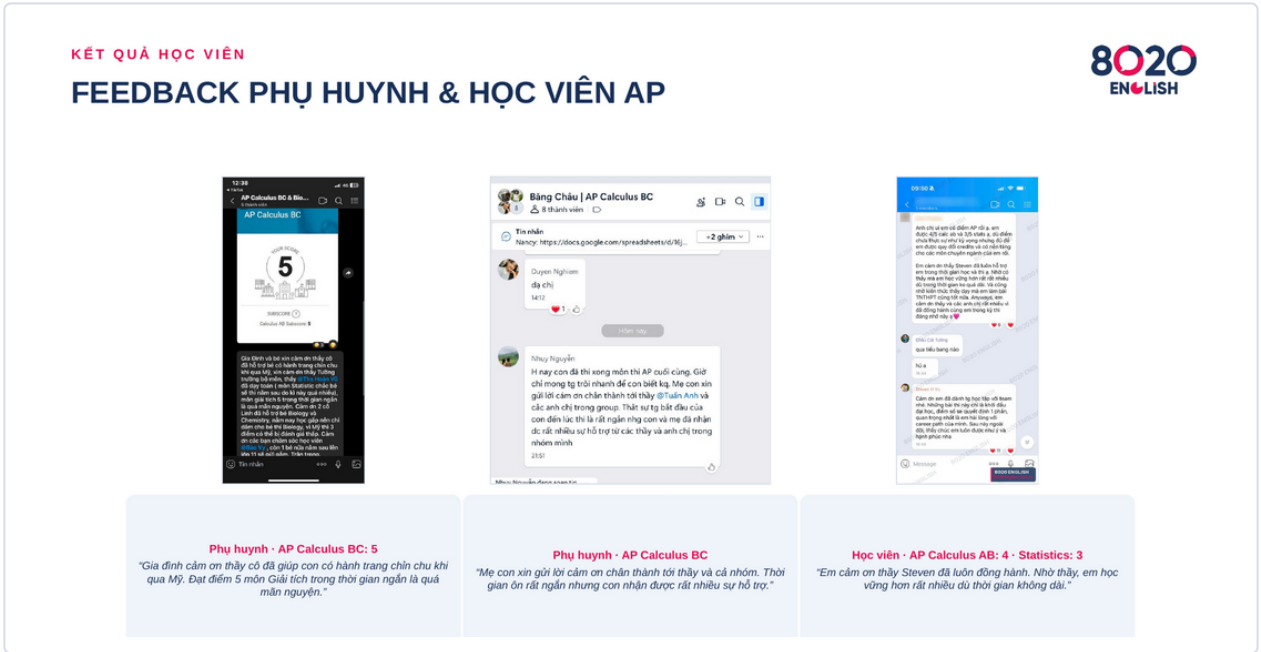
Cảm nhận học viên & phụ huynh

> *"Bạn đã cải thiện được tư duy Writing — kỹ năng từng là nỗi sợ lớn nhất — và cuối cùng đã đậu vào trường top như mong muốn. Thái độ học tập cực kỳ nghiêm túc; ngay cả khi nằm viện vẫn cố gắng học đều đặn."*
>
> — Phan Thị Thanh Hằng · IELTS Overall 8.5

> *"Cần tất cả kỹ năng trên 7.5 để xét vào trường top 1 Anh Quốc về Y học (chương trình UKFPO của NHS) — và đã đạt mục tiêu sau khi học Writing 1–1."*
>
> — Trần Hoàng Bảo Ngọc · IELTS Overall 8.0 (Writing 6.0 → 7.5)

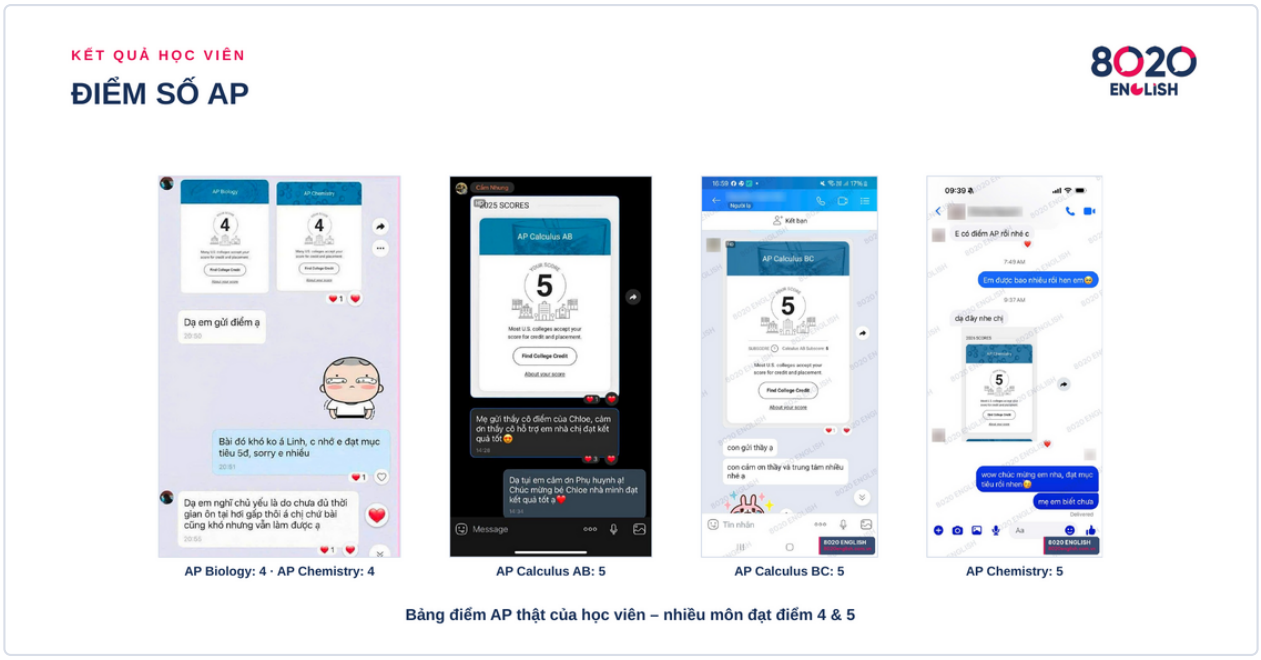
> *"Gia đình và bé xin cảm ơn thầy cô đã giúp con có hành trang chỉn chu khi qua Mỹ. Đạt điểm 5 môn Giải tích trong thời gian ngắn là quá mãn nguyện."*
>
> — Phụ huynh · AP Calculus BC 5/5

> *"Em cảm ơn thầy đã luôn đồng hành. Nhờ thầy, em học vững hơn rất nhiều dù thời gian ôn không dài."*
>
> — Học viên · AP Calculus AB 4

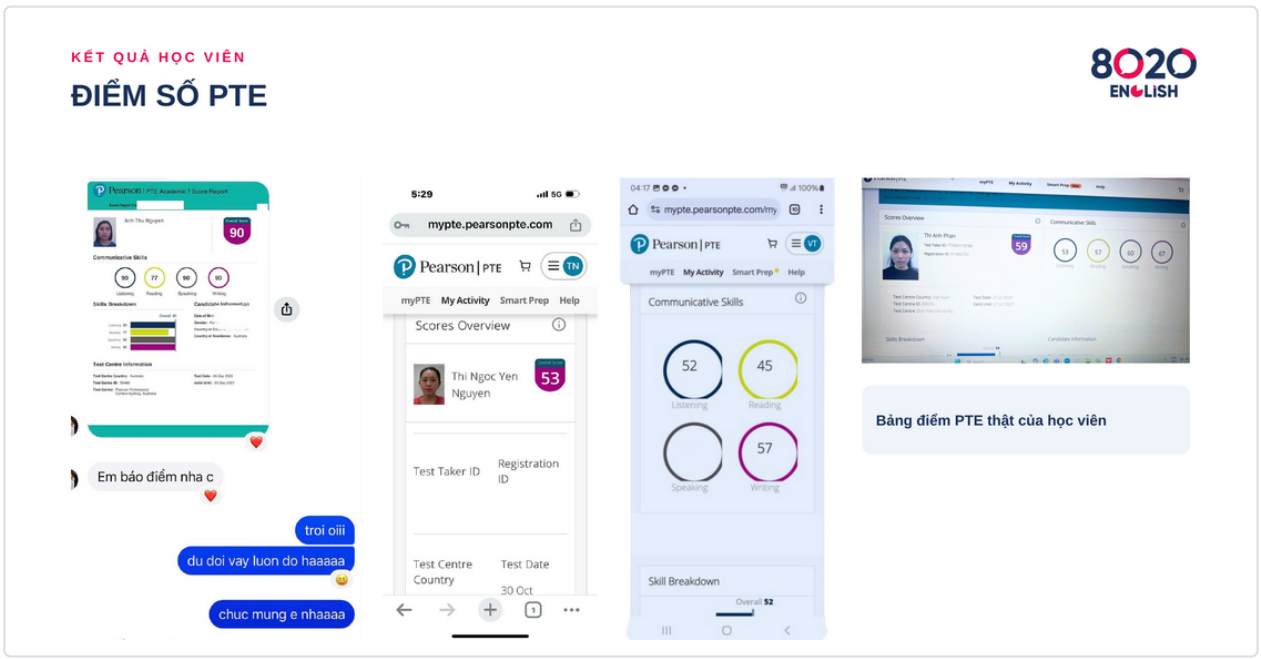


Feedback phụ huynh & học viên AP — nhiều môn đạt điểm 4 & 5

Điểm thi thật của học viên



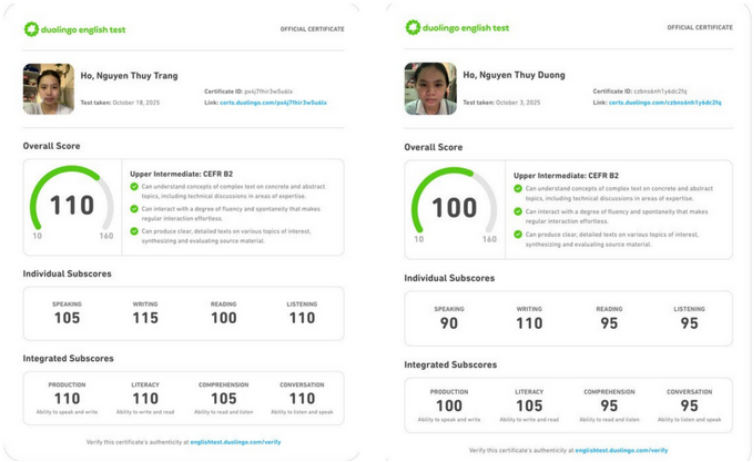
Bảng điểm AP thật — Calculus AB/BC 5/5 · Biology & Chemistry 4–5 (College Board)



Học viên 8020 — PTE Academic 90

KẾT QUẢ HỌC VIÊN

ĐIỂM SỐ DUOLINGO



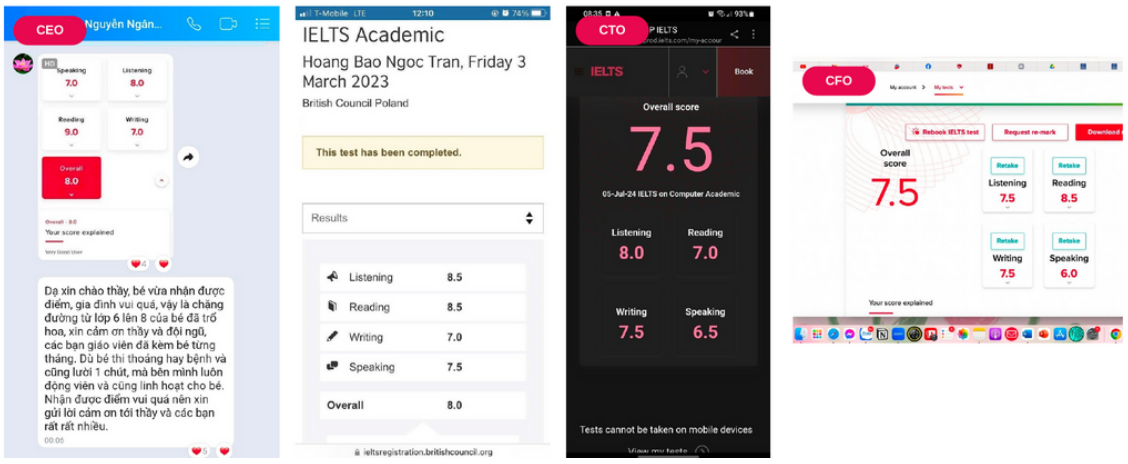
Duolingo English Test – 110 & 100

Học viên 8020 — Duolingo English Test 110 & 100

KẾT QUẢ HỌC VIÊN

THÀNH TÍCH HỌC VIÊN

Bảng điểm thi thật – chất lượng được giám sát nghiêm ngặt. Một số học viên chỉ cần 1–2 khóa để đạt kết quả mong muốn.



Học viên 8020 — IELTS 8.0 / 7.5 / 8.5 (báo điểm thật)